N82-29022

RTI/2094/02-02F

NASA Contractor Report 165926

PROBLEMS RELATED TO THE INTEGRATION OF FAULT-TOLERANT AIRCRAFT ELECTRONIC SYSTEMS

Contract Number NAS1-16489

NASA
National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

June 1982

NASA Contractor Report 165926

PROBLEMS RELATED TO THE INTEGRATION OF FAULT-TOLERANT AIRCRAFT ELECTRONIC SYSTEMS

J. A. Bannister, K. Trivedi,
V. Adlakha, and T. A. Alspaugh, Jr.

Research Triangle Institute
Research Triangle Park, North Carolina

Contract NAS1-16489
June 1982

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

PREFACE

This report was prepared by the Research Triangle Institute, Research Triangle Park, North Carolina, for the National Aeronautics and Space Administration under Task 2 of Contract No. NAS1-16489. The research was conducted under the direction of personnel in the Flight Electronics Division, Langley Research Center. Mr. Milton Holt was the Langley Technical Representative for this task.

Development of the methodology has been a team effort, with significant contributions made by Milton Holt and Rick Butler of Langley Research Center.

Joseph A. Bannister was the RTI Project Manager for the study.

The authors of this report are:

J. A. Bannister

K. Trivedi

V. Adlakha

T. A. Alspaugh, Jr.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF TABLES

1.0 INTRODUCTION

In recent years, electronics in commercial transport aircraft have become increasingly sophisticated as the functions of flight control, guidance, navigation, propulsion control and communication have steadily been enhanced through the use of programmed digital hardware. A continuation of this trend is reflected in the desire to fully integrate aircraft electronic functions. This report is directed at the identification of a methodology for the design of systems that fully integrate the aircraft electronic functions mentioned above.

System integration is measured by the ability of separate tasks to share system resources and exchange information. Past aircraft electronic systems had a low level of integration. A typical system as shown in Figure 1.1, had the following salient system characteristics:

- There was a proliferation of dedicated, independent processors.
- Each processor viewed itself as the center of the universe. There was no communication between processors.
- Tasks were rigidly segregated. For instance, flight control functions and navigation functions were performed by physically separate subsystems.
- Since processors did not communicate, algorithms and data structures could not exploit concurrency.
- The lack of hardware redundancy made it necessary to use mechanical backup.
- Analog technology permeated the system.

Current aircraft electronic systems reflect an increase in integration, as typified by the aircraft electronics of the Boeing 757. A functional description of this system is illustrated in Figure 1.2. Characteristics include:

- Specialized processors are dedicated to specific tasks.
- Interprocessor communication is limited to a restricted subset of processors.

PAST AIRCRAFT ELECTRONIC SYSTEM

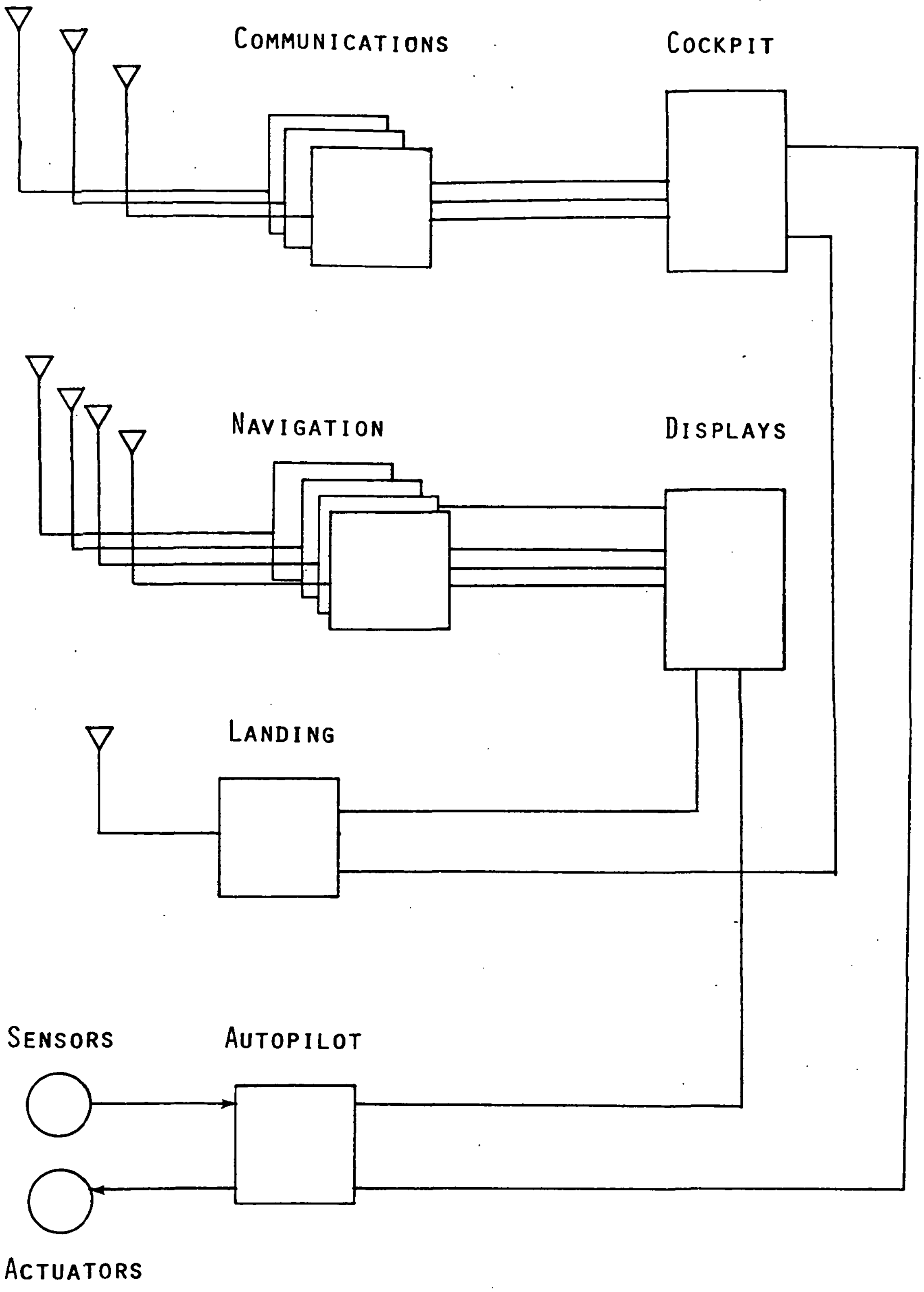
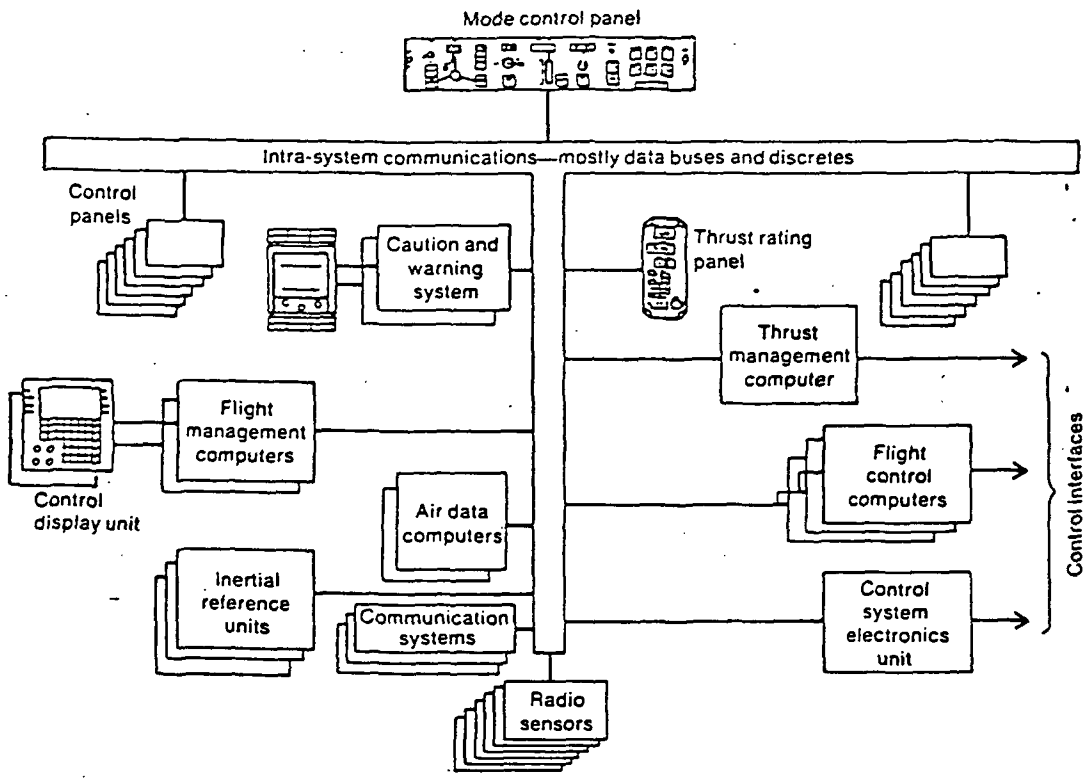


Figure 1.1

CURRENT AIRCRAFT ELECTRONIC SYSTEM



BOEING 757

Figure 1.2

- Tasks are segregated.

- Some algorithms cooperate to take advantage of concurrency. However, the number of cooperating algorithms is still far below potential.

- The use of hardware redundancy is ad hoc rather than systematic. Mechanical backup is still necessary.

- Extensive analog technology is still present.

One can extrapolate the historical tendency of aircraft electronics toward integrated systems. The functional elements of a postulated future system are shown in Figure 1.3. Some of its characteristics are:

- Processors are general-purpose rather than specialized. This enhances the modularity and flexibility of the system.

- There is full interprocessor communication. This permits increased information utilization by processors. It also allows the system to globally monitor itself.

- Tasks are fully integrated. Any processor can perform any task. This increases system flexibility and makes possible reconfiguration and graceful degradation.

- Full interprocessor communication, processor modularity and task integration make it possible to fully exploit the power of concurrency.

- A systematic use of hardware redundancy exists in order to achieve ultrahigh reliability.

- The trend is toward fully digital processing elements.

A matrix comparing the characteristics of past, present and future systems is shown in Table 1.1.

The remainder of this report will consider the establishment of a methodology to identify and design fault-tolerant integrated aircraft electronic systems. This study proceeds by considering the design of the hardware for an integrated aircraft electronic system, and questions relating to the programming and control of the system.

FUTURE INTEGRATED AIRCRAFT ELECTRONIC SYSTEM

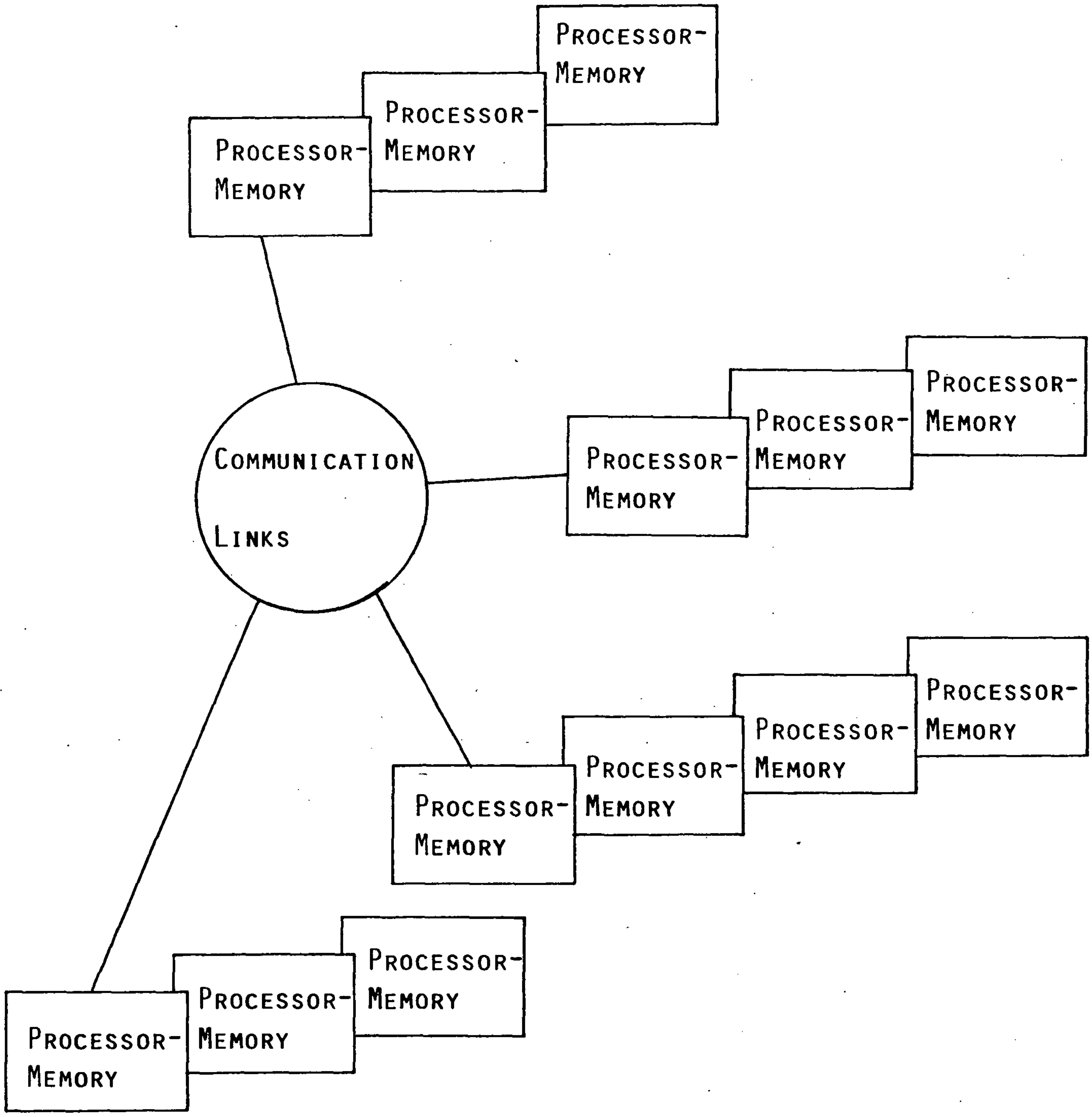


Figure 1.3

Table 1.1. Summary of System Characteristics

System	Modularity	Fault Tolerance	Cooperation	Localization	Concurrency	Technology
Past	Special-Purpose Processors	Achieved by Mechanical Backup	Absent	Each Task Localized to a Processor	Blind	Analog
Present	Special-Purpose Processors	Achieved by Limited Hdwe. Redundancy + Mechanical Backup	Some Tasks Cooperate	Each Task Localized to a Processor	Myopic	Hybrid Digital-Analog
Future	General-Purpose Processors	Achieved by Systematic Use of Hdwe. Redundancy	Cooperation Possible Among Arbitrary Sets of Tasks	Any Processor May Perform Any Task	Global	Digital

2.0 DESIGN OF THE HARDWARE FOR AN INTEGRATED AIRCRAFT ELECTRONIC SYSTEM

In general, programmed systems such as integrated aircraft electronic systems can be divided into two major components: hardware and software. The design of a system's software traditionally follows the design of its hardware, since the activity of programming a system depends upon the pre-existence of the system. In this chapter the design of the hardware component of the system is discussed.

The system hardware design will be discussed first in terms of a taxonomy of systems that are appropriate for program-controlled aircraft electronics. A taxonomy is a powerful tool in system design since it allows one to systematically categorize a potentially unbounded number of possible system designs. Having thus categorized the designs, one is in the position to determine which systems are suitable for the intended application. Thus, it is possible to select a specific system. After establishing the taxonomy, a methodology for choosing a suitable hardware structure for the system will be considered. Having done this, the methodology will be supplied to select a candidate integrated avionic system.

2.1 A Taxonomy of Computing Systems Capable of Concurrent Instruction Execution

The purpose of this section is to propose a taxonomy of computing systems that are capable of concurrent instruction execution. Such systems are often referred to as concurrent, parallel or multiprocessor systems.

This section effectively ignores nonconcurrent, nonparallel or uniprocessor systems for several reasons. First, in the aircraft electronics suite, a large number of functions must be simultaneously performed. This suggests that a computing system that performs these functions must be capable of simultaneous and independent instruction execution. It is unlikely that a nonconcurrent system could be time-shared by these functions in any effective manner. Second, since avionics perform extremely critical functions, there is a driving need to ensure the reliability of the system. Fault-tolerant methods are the proven approach to achieve high reliability.

These methods rely on hardware redundancy to tolerate faults in the system. Thus, system modules, whose function it is to interpret and execute program instructions, are replicated. By virtue of this redundancy, such a system is capable of independent, concurrent instruction execution. One can see, therefore, that a concurrent system is the only reasonable alternative for any candidate integrated avionic system. Thus, the taxonomy will deal exclusively with concurrent systems.

2.1.1 Motivations for the use of multiprocessor systems. - The power of multiprocessor systems is derived from their ability to achieve algorithmic speedup over uniprocessor systems. A multiprocessor system is understood to be any programmed system of two or more processors capable of independent, simultaneous instruction execution and information exchange via some interconnection mechanism (See ref. 30).

In addition to the high speed capability, there are many reasons (See ref. 30) for employing multiprocessor systems in integrated flight electronic systems. These reasons include:

Load sharing. In the event that one processor has a heavier load than the other processors in the system, some of its load can be redistributed to the other processors.

Peak computing power. It is possible to devote the entire system to a single task.

Performance/cost. There is an abundance of small processors with an instructions/second/dollar ratio superior to that of large "super processors."

Graceful degradation. One can design multiprocessor systems with no central critical components. Failures may then be configured out of the system and the remaining processors may take up all or part of the failed unit's load.

Modular growth. One can design systems in which processors, memories and I/O subsystems are added incrementally.

Functional specialization. One can add functionally specialized processors to improve performance for a particular application.

2.1.2 Taxonomic framework. - The classical paradigm for taxonomies is the Linnaean taxonomy that is used for biological classification. Two major principles of a taxonomic framework are illustrated by the Linnaean taxonomy:

1) A good taxonomy is hierarchical. That is, the subject being taxonomized can be partitioned from a very coarse level down to a very fine level. For instance, the Linnaean taxonomy dichotomizes all life into plant and animal kingdoms; animals are further divided into vertebrates and invertebrates, etc. This taxonomy forms an eight-level tree structure with gradations of resolution ranging from kingdoms (coarse) to species (fine).
2) A good taxonomy is based on the functional properties of the objects being classified. For example, a taxonomy based solely on a machine's physical structure is unlikely to be very useful (See ref. 8). Thus, a taxonomy of multiprocessor systems that integrates structural and functional system properties in its characterization of these systems should be considered.

A taxonomy of multiprocessor systems should embody these two important principles.

Although a taxonomy for multiprocessor systems is desired, in reality very few multiprocessor systems have been designed. Even fewer systems have been built and a mere handful of relatively unsophisticated multiprocessor systems are commercially available. Given such a paucity of practical design experience in the field of multiprocessor systems, it is necessary to consider a taxonomy for potential as well as actual systems.

Many decisions must be made when designing a multiprocessor system. The totality of these decisions will be referred to as the "multiprocessor system design space." Equivalently, the term "design space" will mean the complete collection of actual multiprocessor system designs.

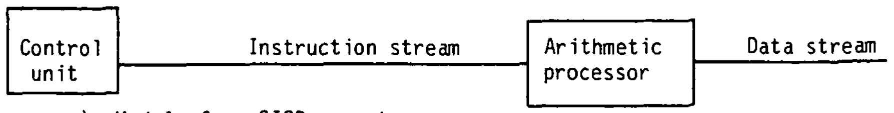
2.1.3 Previous taxonomies of multiprocessor systems. - Several taxonomies of the multiprocessor system design space exist. Four such taxonomies, Flynn's (See ref. 12), Anderson and Jensen's (See ref. 1), Davis, Denny and Sutherland's (See ref. 8), and Siewiorek's (See ref. 30), are discussed below.

2.1.3.1 Flynn's taxonomy: Flynn's taxonomy is a well-known taxonomy that is based on the multiplicity of data and instruction streams. There are only four classes in this taxonomy, as shown in Figure 2.1. They are SISD, SIMD, MISD and MIMD, which stand for single instruction (stream), single data (stream); single instruction (stream), multiple data (stream); multiple instruction (stream), single data (stream); and multiple instruction (stream), multiple data (stream), respectively. Models of each of these classes are shown in Figure 2.2.

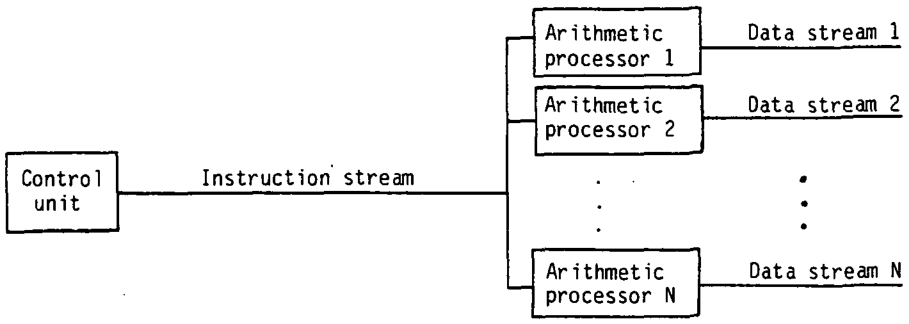
Flynn's taxonomy certainly integrates both structural and functional properties of multiprocessor systems. His classification of these systems is based on extremely important system features, viz., control and storage features.

However, the taxonomy is not based upon a hierarchical organization. There is only one level of classification, which leaves much to be desired when one is considering the design of a system. Essentially, the only design parameters that can be controlled explicitly are the multiplicities of the instruction stream or of the data stream. Clearly, much more is required for a taxonomy of multiprocessor systems. A further drawback of this taxonomy is that the resolution of its classes is far too coarse. There are only four basic classes in the design space and each class may comprise an unmanageable number of system designs.

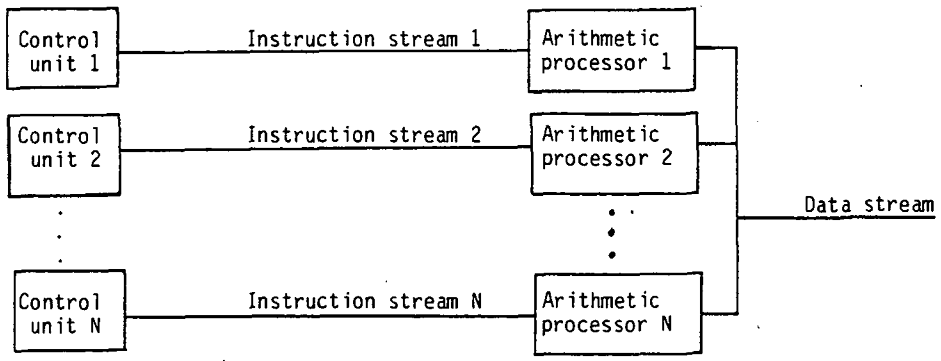
One goal of a design methodology is the ability to choose a design that will fit the intended system application. It is desirable that, once the functional and operational requirements of a system have been identified, there will be a methodology that is capable of selecting suitable designs for the system. Flynn's taxonomy refers only to instruction and data channels. It is doubtful that one can relate these two design parameters to the given requirements in a meaningful way. Certainly, nothing is known in the general literature that indicates how to relate these elements. This taxonomy is, therefore, unsuitable for use in a design methodology for integrated aircraft electronic systems.



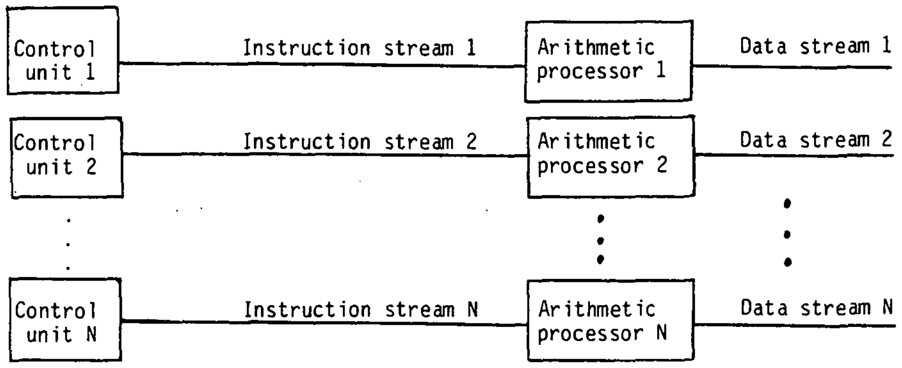
a) Model of an SISD computer



b) Model of an SIMD computer



c) Model of an MISD computer



d) Model of an MIMD computer

Figure 2.1. Models of Computer Systems.

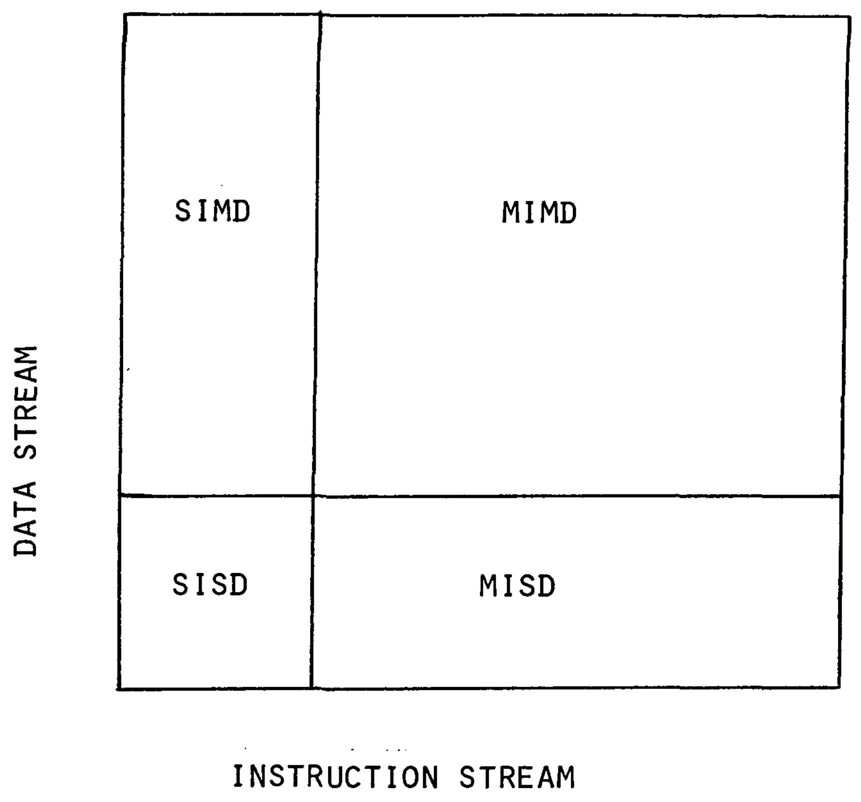


Figure 2.2. Flynn's Partitioning of the Design Space.

2.1.3.2 Anderson and Jensen's taxonomy: The taxonomy of Anderson and Jensen is primarily directed toward the classification of multiprocessor system interconnection structures. This taxonomy assumes three primitive notions--the processing element (PE), which is any unit that is able to execute instructions, and the paths and switching elements, which make up the interconnection structure. A path is a medium through which messages can be transferred between other system elements, e.g., wires, radio, common-carrier links. A switching element is an entity that determines the path that a message will take, e.g., elements for routing and dispatching. The taxonomy describes the possible configurations of the three architectural primitives: PE's, paths and switching elements.

This taxonomy, shown in Figure 2.3, is clearly hierarchically organized. The premise of this organization is that the multiprocessor design space is considered to be a tree. Designing a system is tantamount to making a series of design decisions. In the taxonomy of Anderson and Jensen there are four basic design decisions to be made:

1) transfer strategy
2) transfer control method
3) transfer path structure
4) system architecture

Decisions (1) and (2) are strategic decisions; i.e., they involve very fundamental policy decisions. Such decisions usually have a far-reaching effect on the system's operational capabilities and may have a significant impact on such important system features as performance, reliability, cost effectiveness and modularity. The first decision is whether to use direct or indirect transmission of messages from a source to a destination. Indirect transmission is distinguished from direct transmission by the presence of message-switching entities between sources and destinations. Units such as repeaters and temporary buffers do not constitute indirect communication mechanisms; rather they are to be considered integral parts of the direct link. Nor do preprocessing or postprocessing decisions applied by source or destination elements to messages imply indirection. In the event that an indirect transfer strategy has been chosen, the

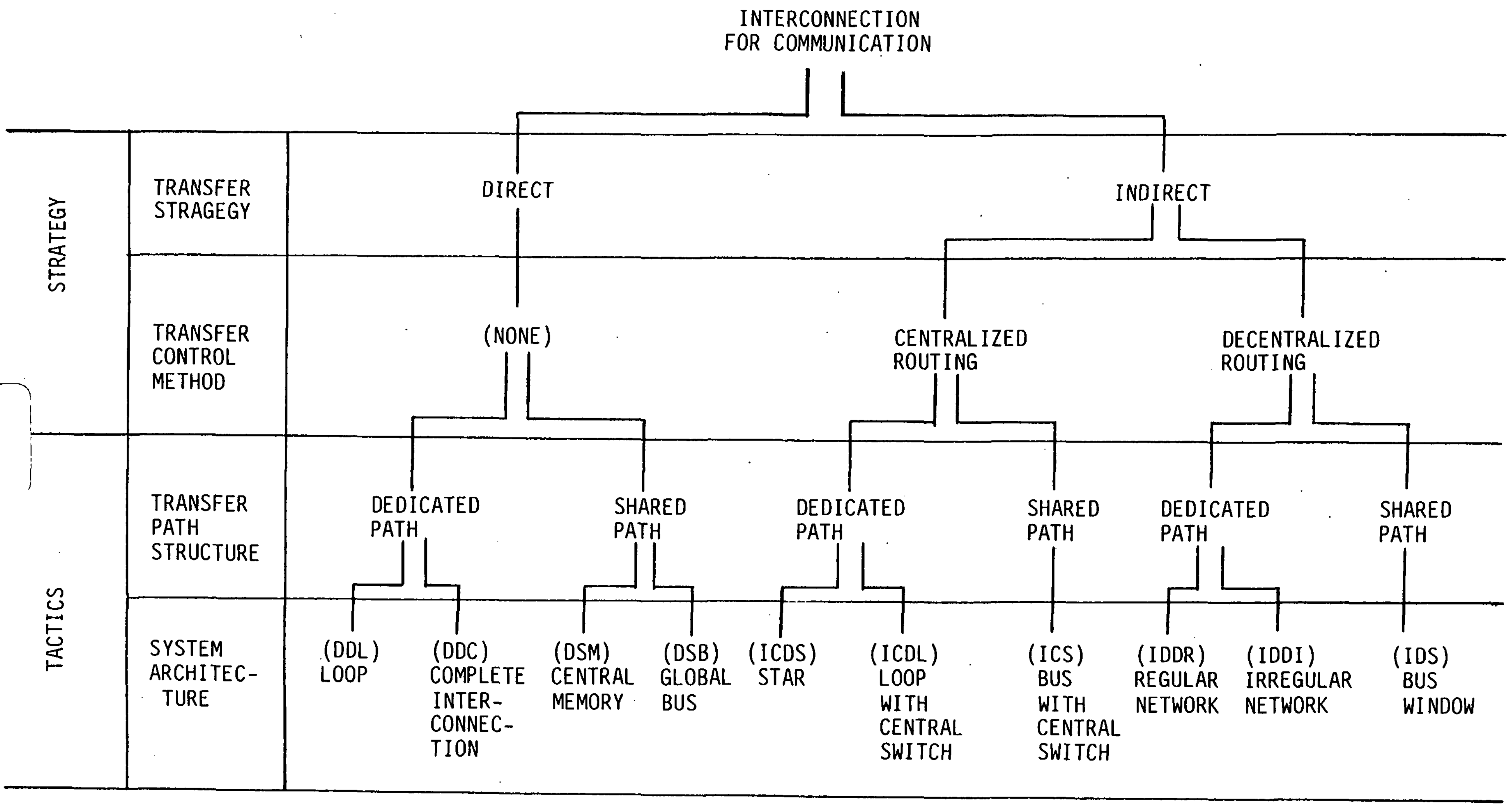


Figure 2.3. An Interconnected Computer System Is the Result of a Series of Design Decisions, and the Decision Space Can Be Considered to Be a Tree.

designer must also decide whether the switching of messages is to be effected by a sole entity (centralized routing of messages) or by several entities (decentralized routing of messages).

Design decisions (3) and (4) are tactical decisions in that they concern themselves with what are commonly known as implementation details. Decision (3) determines the structure of the transfer path. Two alternatives exist for the transfer path structure: the path may be dedicated or shared. Decision (4) provides for the choice of the system topology. Anderson and Jensen identified 10 basic topologies in their taxonomy:

1) Loop (DDL): Figure 2.4
2) Complete interconnection (DDC): Figure 2.5
3) Central memory (DSM): Figure 2.6
4) Global bus (DSB): Figure 2.7
5) Star (ICDS): Figure 2.8
6) Loop with central switch (ICDL): Figure 2.9
7) Bus with central switch (ICS): Figure 2.10
8) Regular network (IDDR): Figure 2.11
9) Irregular network (IDDI): Figure 2.12
10) Bus window (IDS): Figure 2.13

(The acronyms denote the path on the taxonomy tree, which begins at the root and ends at the leaf that describes the system under consideration. For example, DDL stands for Direct Dedicated Loop.)

This taxonomy views an interconnected multiprocessor system as a series of design decisions. The design space is then equivalent to the taxonomic tree.

There is a richness in Anderson and Jensen's taxonomy that makes it attractive for real-life uses. It is most definitely a rather complete characterization of interconnection architectures for multiprocessor systems and therefore is of interest to designers of multiprocessor systems.

The major difficulty with the taxonomy is that it suffers from poor resolution. The primitives of the taxonomy are processing elements, paths,

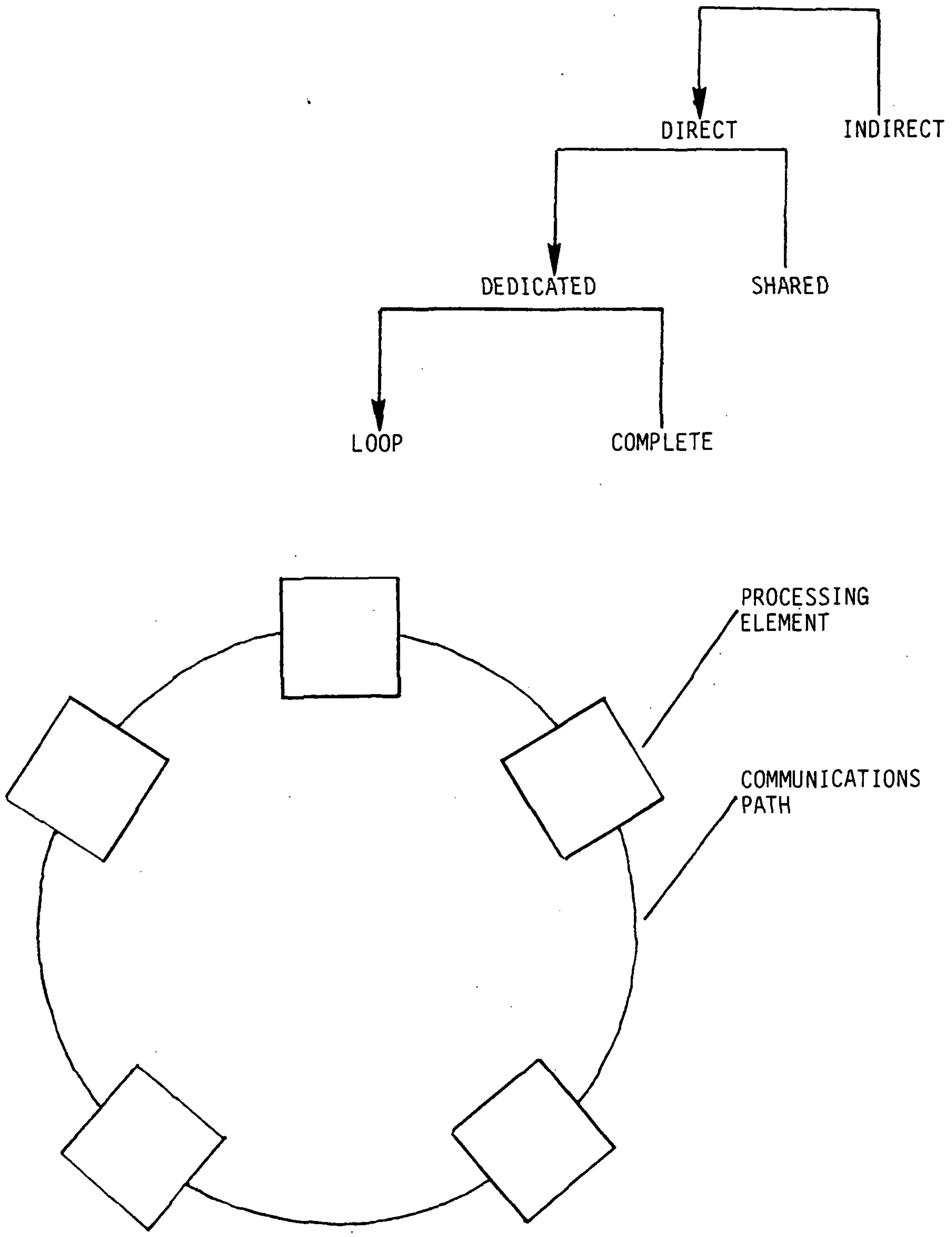


Figure 2.4. DDL (Loop).

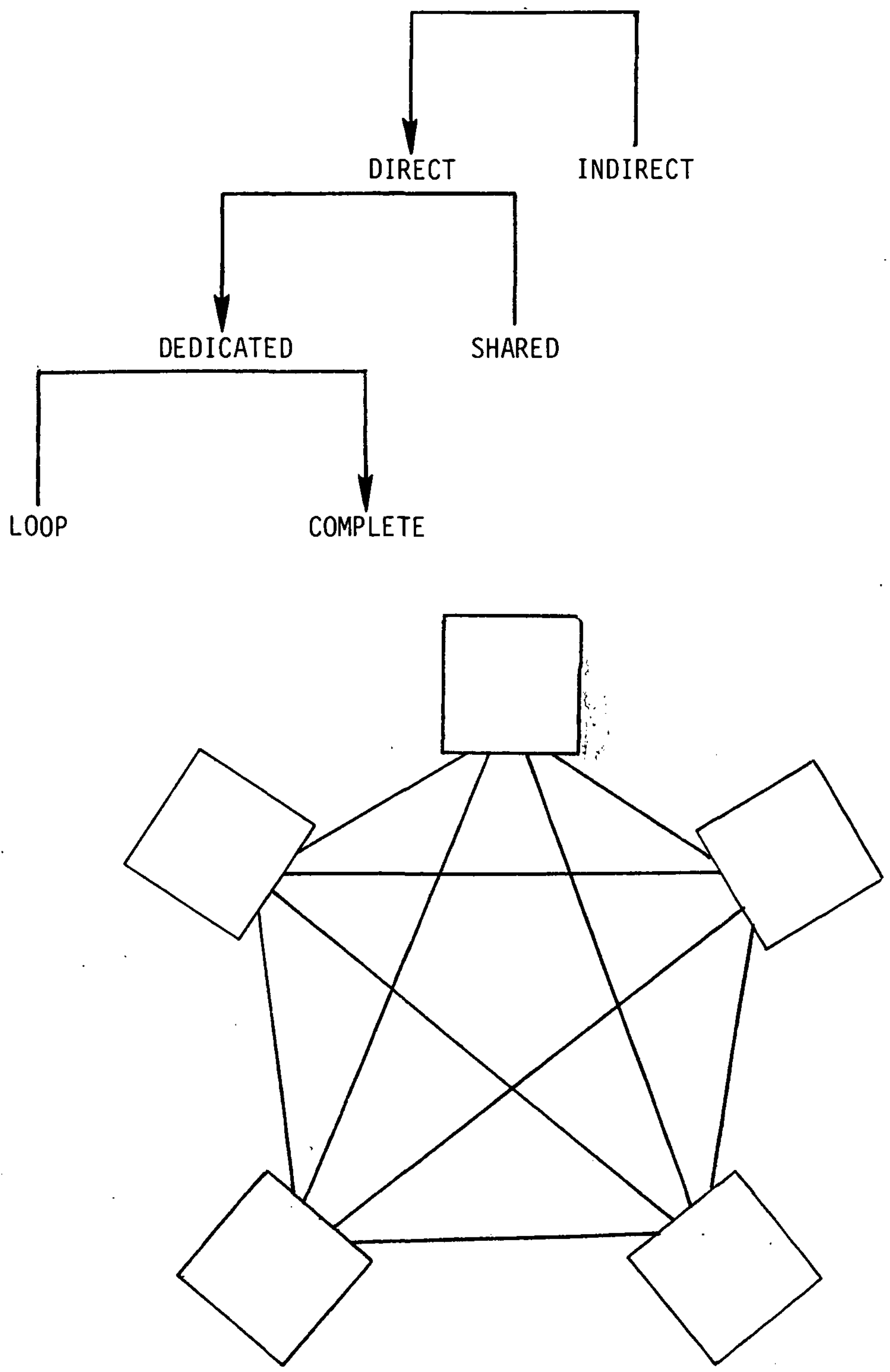


Figure 2.5 DDC (Complete interconnection).

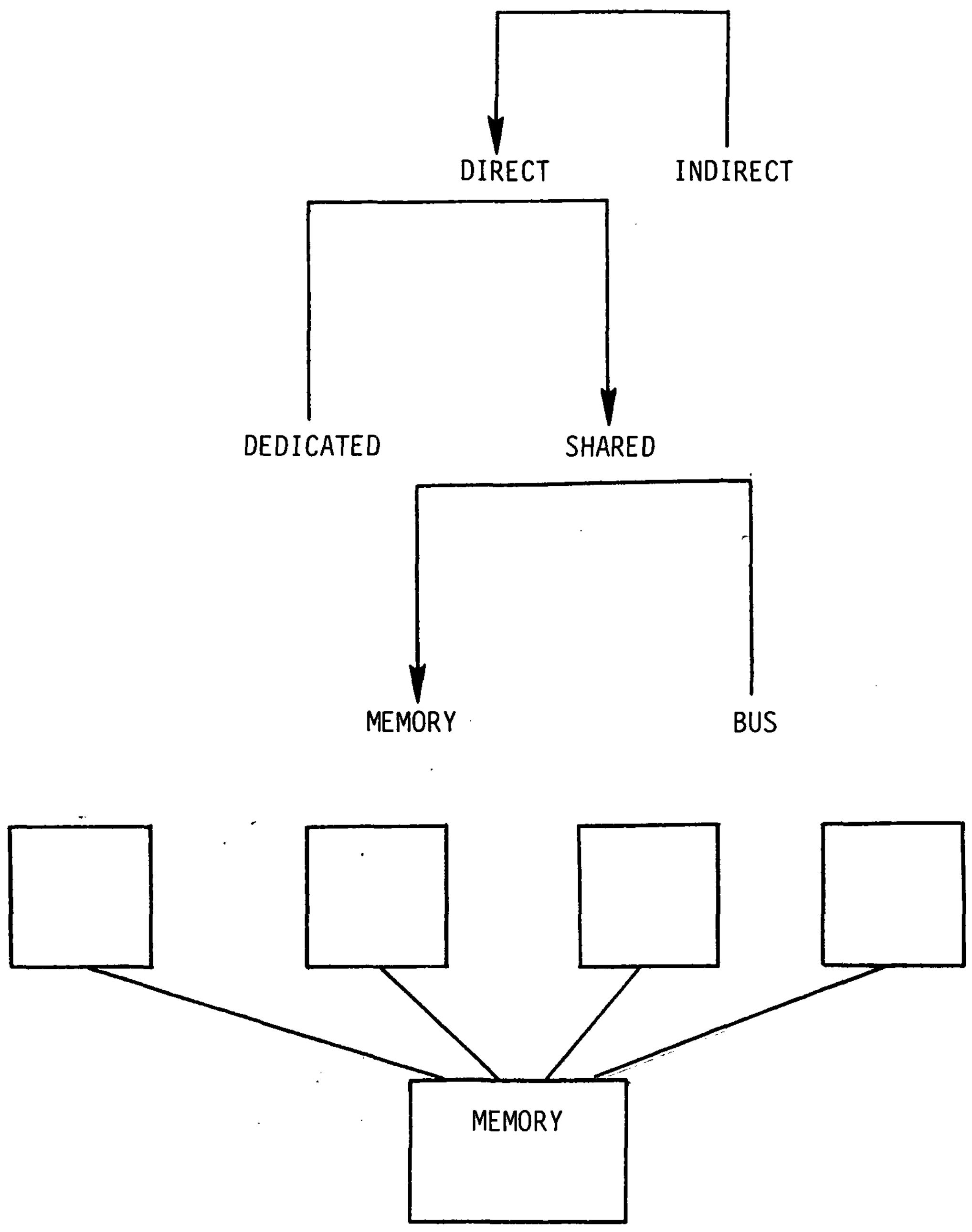


Figure 2.6. DSM (Multiprocessor).

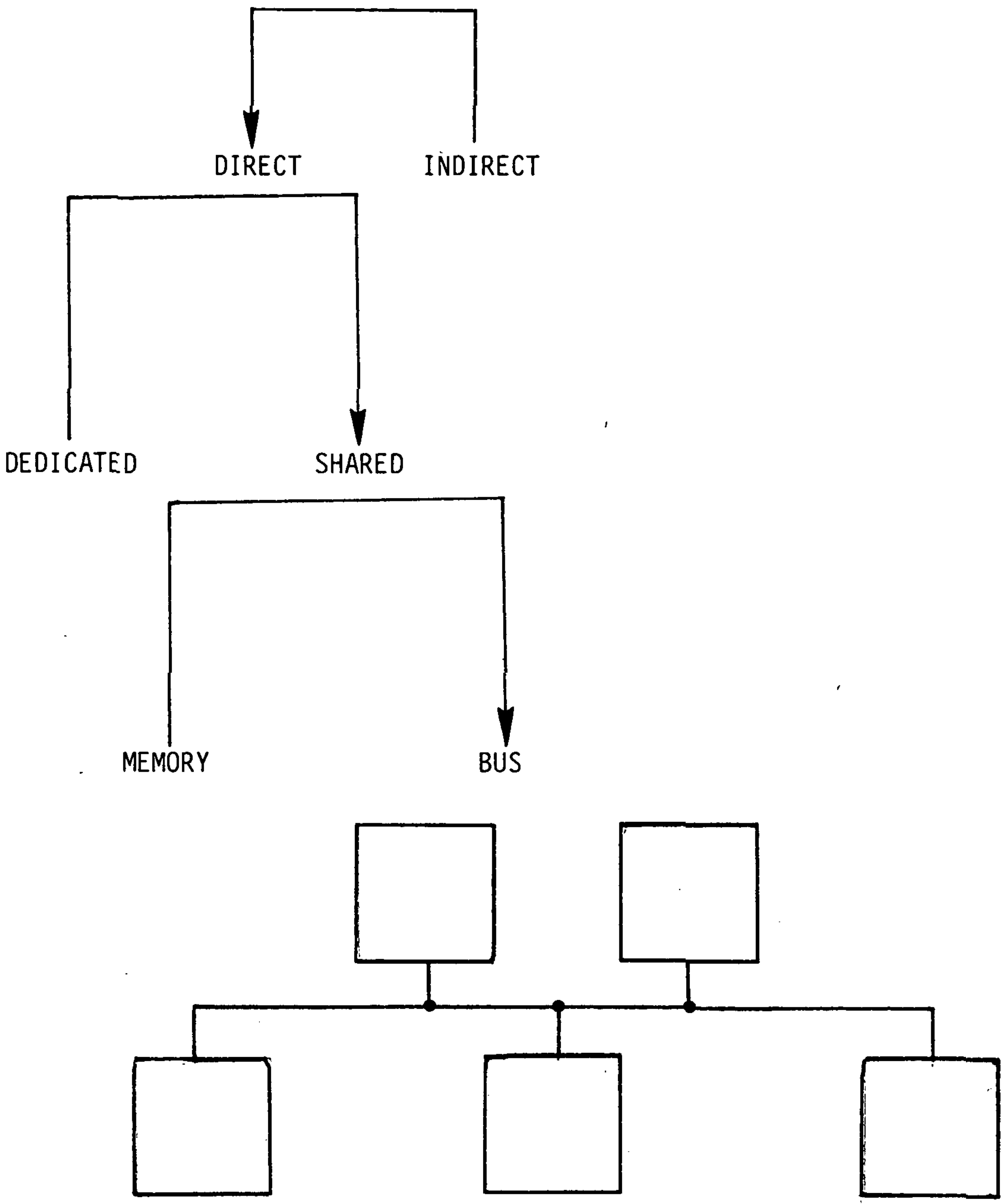


Figure 2.7. DSB (Global bus)

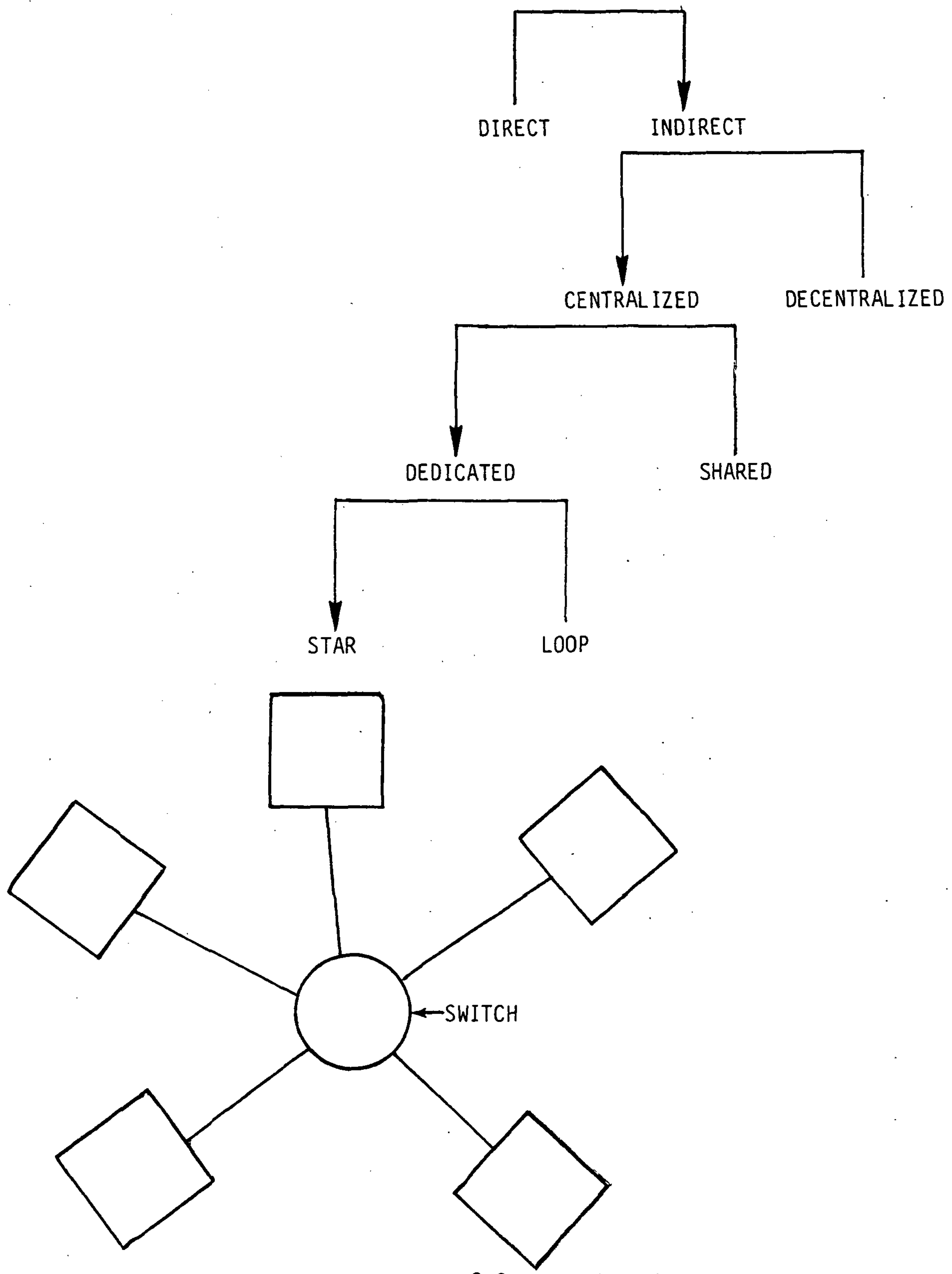


Figure 2.8. ICDS (Star).

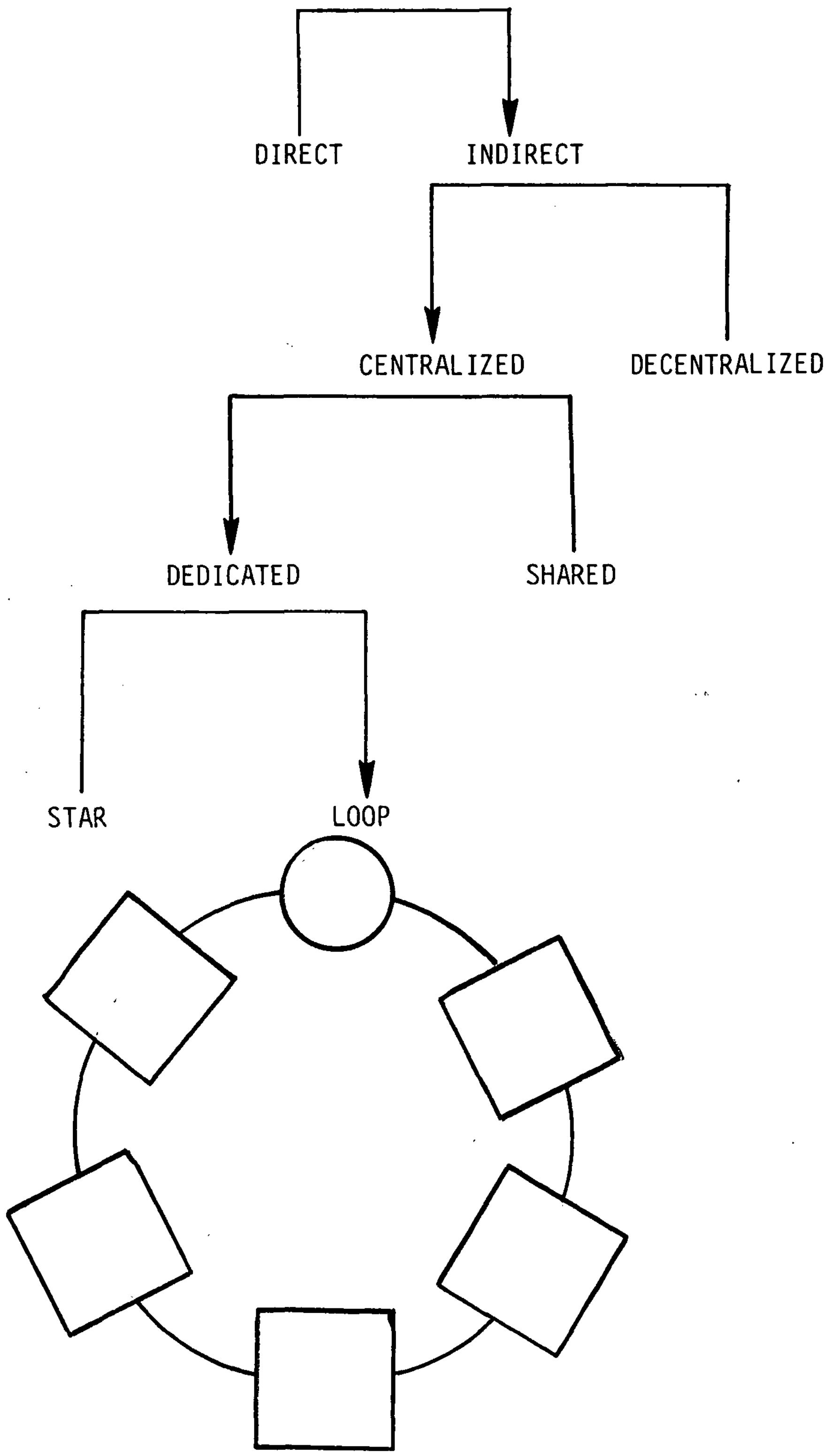


Figure 2.9. ICDL (Loop with central switch).

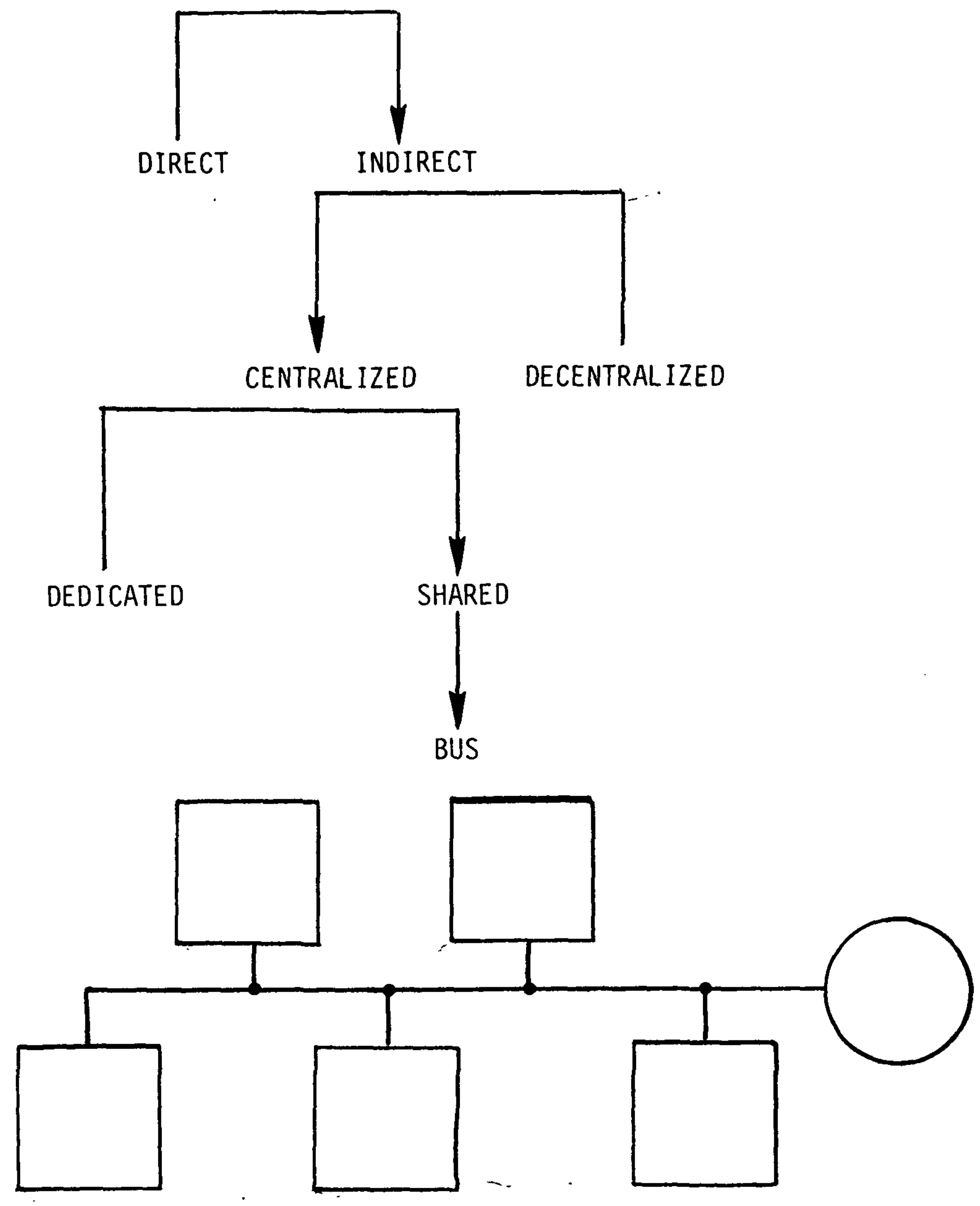


Figure 2.10.. ICS (Bus with central switch).

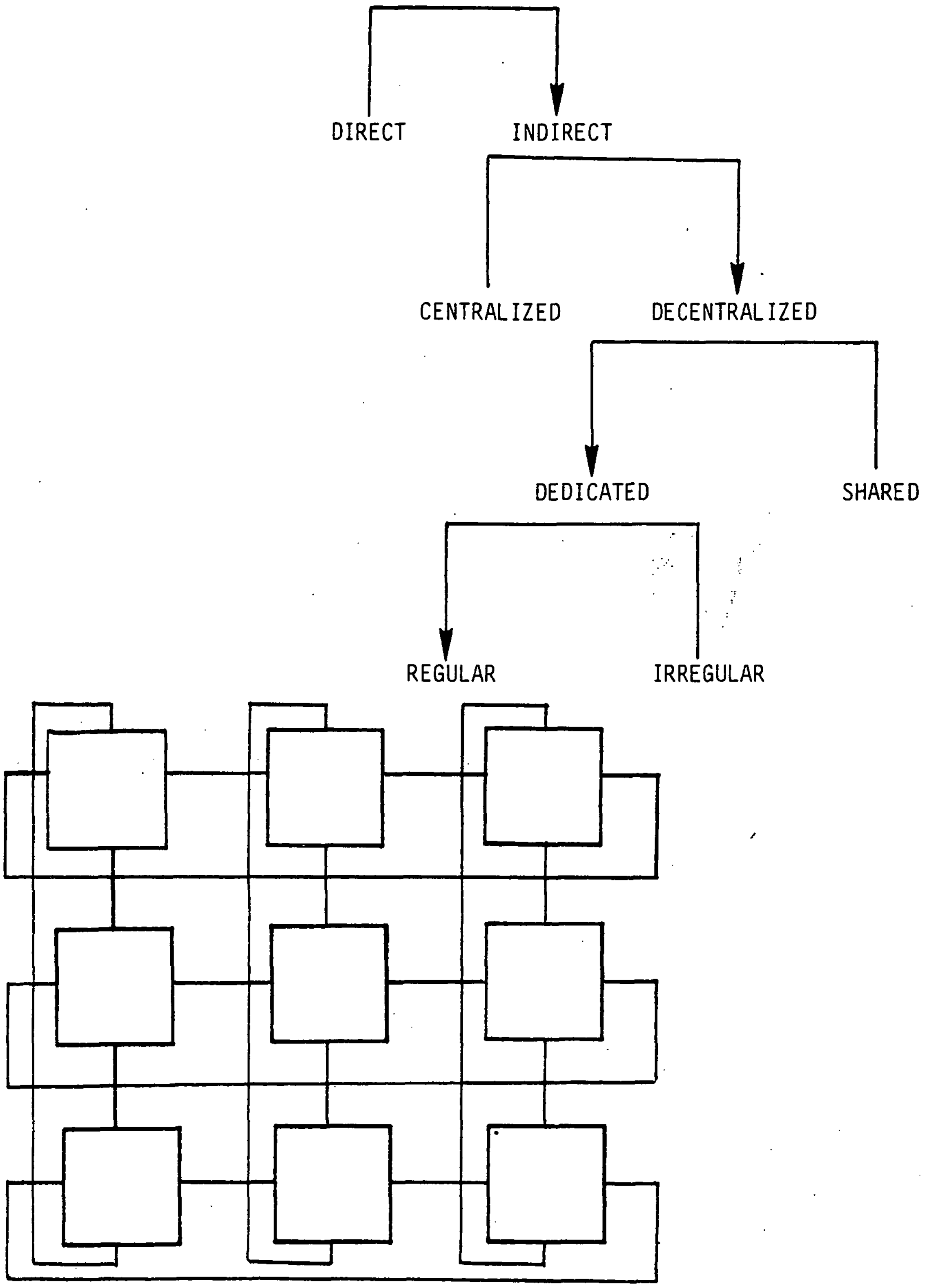


Figure 2.11. IDDR (Regular network).

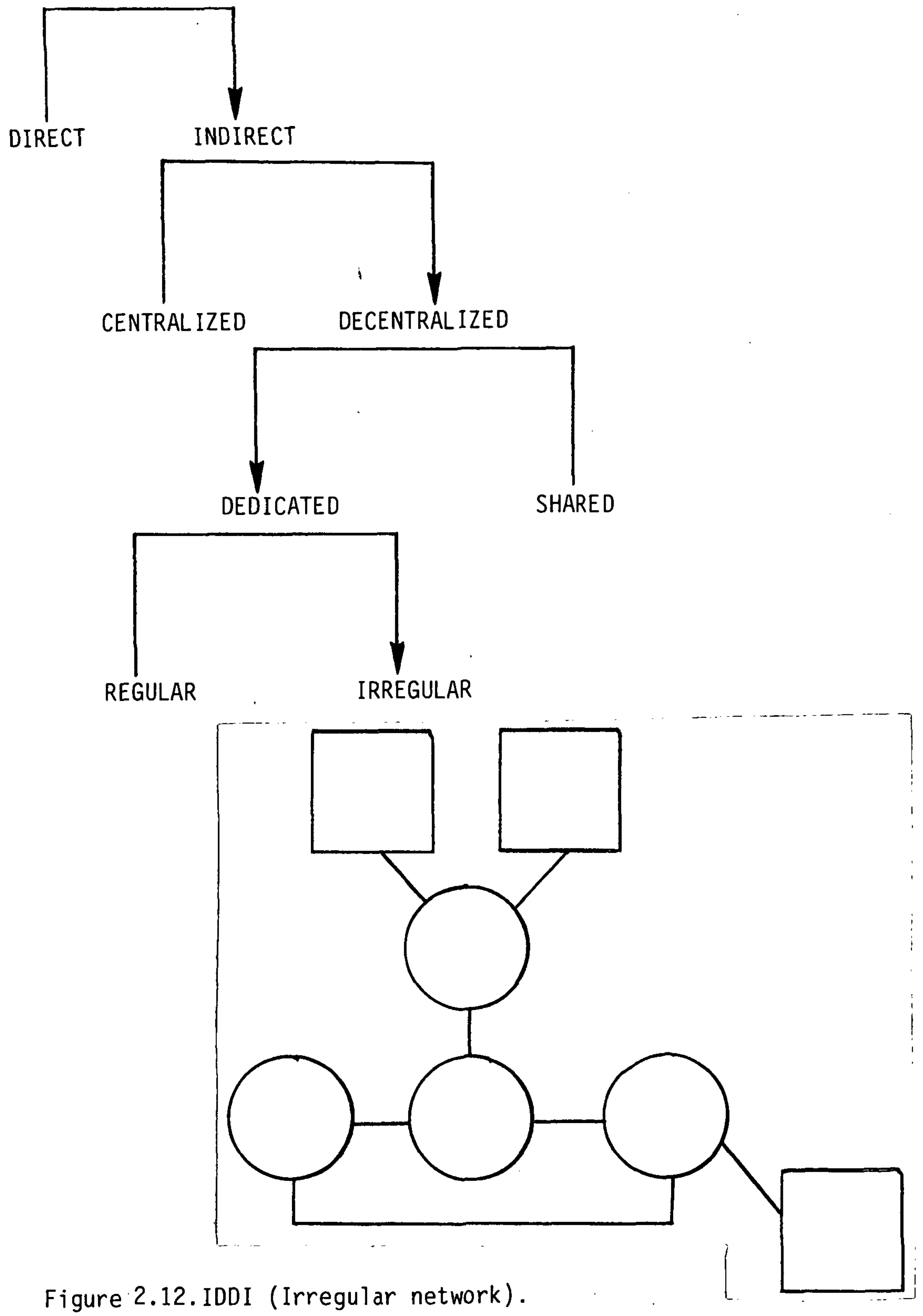


Figure 2.12.IDDI (Irregular network).

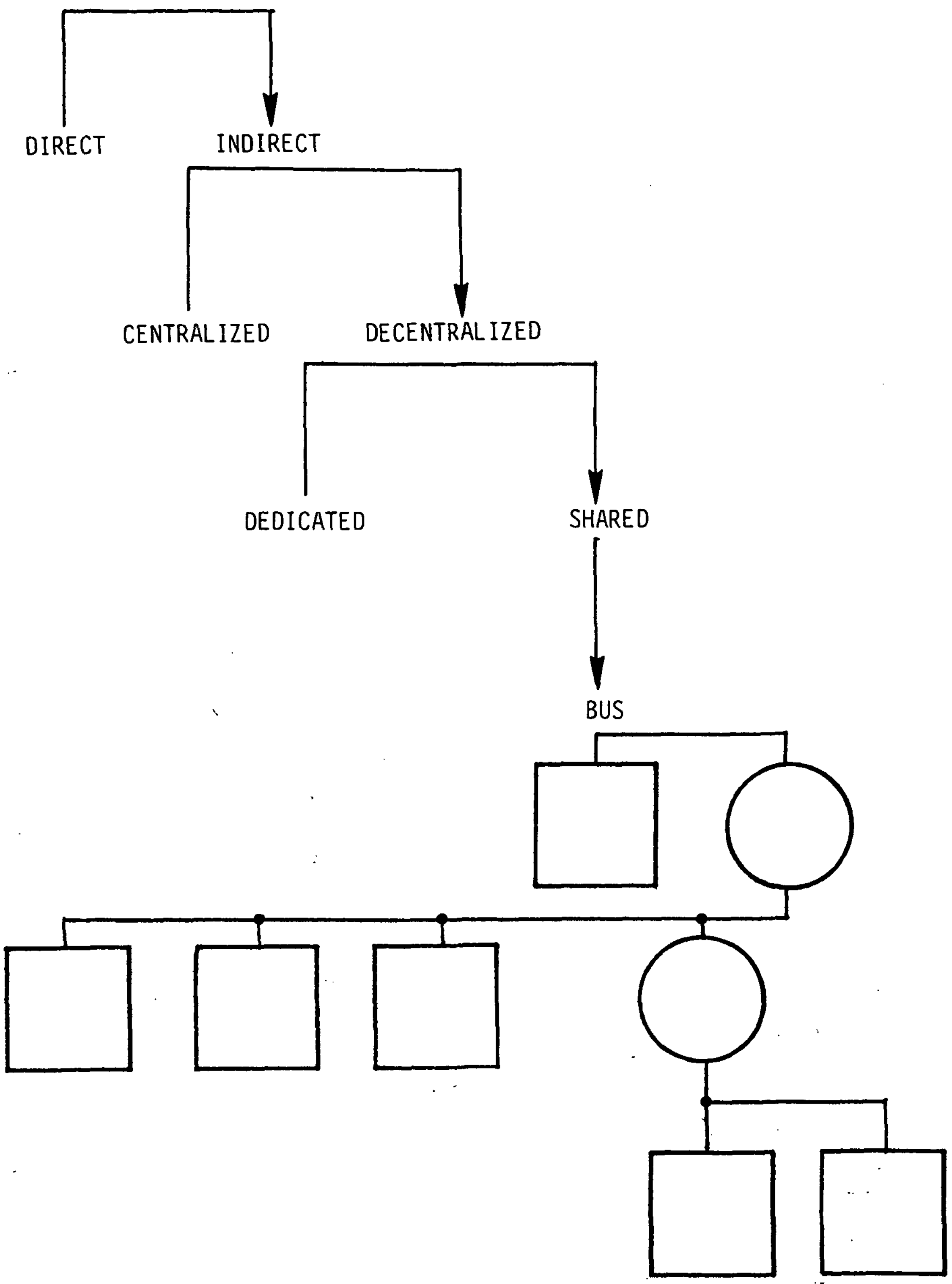


Figure 2.13.IDS (Bus Window).

and switching elements. A designer would prefer a much finer resolution, e.g., details of the memory system, bandwidth ratios, and I/O system structure.

The next taxonomy attempts to reflect the multiprocessor design space at a higher resolution.

2.1.3.3 The taxonomy of Davis, Denny and Sutherland: The taxonomy of Davis, Denny and Sutherland builds upon the taxonomy of Anderson and Jensen in several important ways. The atomic entities of Davis, Denny and Sutherland's taxonomy are paths and elements. A path is considered to be a unit that transmits information without modifying its content; a path does not provide for information storage. An element is an entity that performs an action on data, such as storage or arithmetic operations. As in Anderson and Jensen's taxonomy, there are two basic criteria applied to the atoms: function and structure.

Applying these two criteria yields the taxonomy shown in Figure 2.14. There are four principal taxonomic parameters in this scheme:

1) Topology: path structure
2) Communication: path function
3) Granularity: element structure
4) Operation: element function

Topology, according to Davis, Denny and Sutherland, is the dominant design decision. The interconnection topology parameter is organized exactly as in the taxonomy of Anderson and Jensen.

Communication is the function of the system's paths. Three communication components are considered noteworthy:

1) Mobility: the ratio of program information to data information transferred on the path.

2) ATR: average transmission rate over the path.

3) Bandwidth: the maximum number of bits of information that can be transmitted on the path in 1 second.

	PATH	ELEMENTS
STRUCTURE	TOPOLOGY	GRANULARITY
FUNCTION	COMMUNICATION	OPERATION

Figure 2.14. The function of a path is to provide communication and the path's structure is indicated by its topology. The function of an element is to perform certain operations and the element's most important structural feature is its granularity.

Each of these three components may be quantified to a certain degree. Using a three-level quantification of each component, the communication parameter can be expressed as a cross-product:

$$\text{Communication} = \left\{\begin{array}{l}\text{high bandwidth}\\ \text{medium bandwidth}\\ \text{low bandwidth}\end{array}\right\} \times \left\{\begin{array}{l}\text{heavy mobility}\\ \text{moderate mobility}\\ \text{slight mobility}\end{array}\right\} \times \left\{\begin{array}{l}\text{high ATR}\\ \text{medium ATR}\\ \text{low ATR}\end{array}\right\}$$

The dimension or size of an element needs to be considered. For instance, one must be able to distinguish between a 16 x 16 array of LSI-11's and a 16 x 16 array of CRAY-1's. The size of the largest repeated element is called the system's granularity. It will often suffice to discuss granularity in terms of small, medium and large.

The element's operation is a functional description of the element. The dominant function of an element is the way it transforms input data to output data. An element is also categorized by the ratio of storage devices to combinational logic present in the element--this is called the memory-logic mix of the element. Other aspects of an element's operation are indicated in Figure 2.15.

2.1.3.4 Siewiorek's taxonomy: Siewiorek's taxonomy is concerned with a substantially greater number of design parameters than the previously discussed taxonomies. This is appealing in that it makes it possible for the designer of a multiprocessor system to consider a large number of taxonomically distinct designs. Generally speaking, this taxonomy conveys a great deal of information. It also treats aspects of multiprocessor system design that are ignored by the other taxonomies.

Siewiorek concentrates on seven major design parameters:

1) Node types--processors, memories, switches and other devices.

2) Memory system--a decision parameter that considers the logical address space and the physical organization of memory.

3) Memory switch--a decision parameter specifying the mechanism that provides system components access to shared memory.

4) Processor-memory data paths--a decision parameter that treats properties of data paths such as sharing and bandwidth.

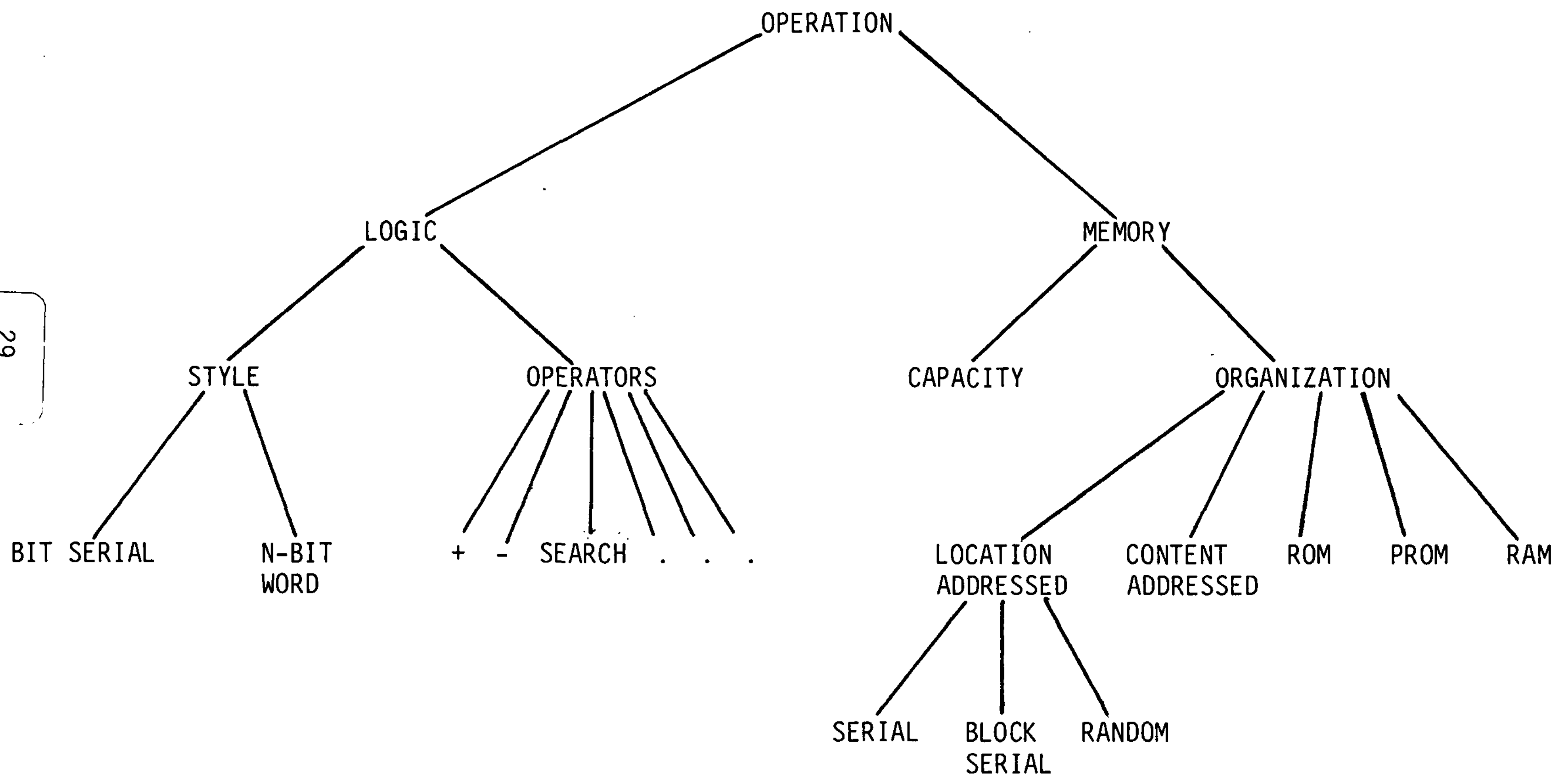


Figure 2.15. Important Aspects of Element Operation.

5) I/O system--a decision parameter that deals with the logical and physical structures of I/O.

6) Ratios--a decision parameter that determines the relationships that other parameters should have to each other. Important ratios include processor-memory and I/O-memory bandwidths.

7) Interprocessor communication--a decision parameter that includes deciding how the hardware will handle interrupts and exceptions.

These decision parameters are further divided into subparameters, which allow the choice of a system to be made in an orderly and simplified manner. For example, the task of selecting the appropriate memory system is decomposed into selecting the logical and physical structures of the memory. Choosing the logical structure is further broken down into choosing among several memory-sharing methods (e.g., local, multiport simplex, shared-overlapped) and choosing among various alternatives for memory protection (e.g., object-based, capability-based). The full taxonomy is displayed in Figure 2.16.

2.1.4 A proposed taxonomy. - Each of the previously considered taxonomies emphasizes certain system characteristics. Two of these--Siewiorek's taxonomy and Anderson and Jensen's taxonomy--will greatly influence a taxonomy for multiprocessor systems. Anderson and Jensen's taxonomy does a particularly thorough job of classifying how multiprocessor systems may be connected. Siewiorek's taxonomy has the advantage of completeness and fineness of resolution--it is capable of examining systems at a fine level of detail. Each of these features is desirable and will be incorporated into the taxonomy for multiprocessor systems.

The ingredient lacking in the previously discussed taxonomies is the explicit consideration of reliability. This ingredient will be included in the taxonomy for multiprocessor systems. Moreover, it will play a central role in the design methodology, in support of which the taxonomy stands. The final taxonomy is illustrated in Figure 2.17.

Dimensions

- Node Types
  - Nonhomogeneous
  - Homogeneous
- Memory System
  - Logical structure of address space
    - Local
    - Shared
      - 1
      - n
      - n(m)
  - Protection
    - None
    - Object
    - Capability
  - Physical structure of memory
    - Size
      - Immediate
      - System
    - Redundancy
      - Replication (*r)
      - Coding
        - Parity (*p)
        - Hamming (*h)
- Memory Switch
  - Logical structure
    - Accessibility
      - All
      - Partial
        - Overlapped
        - Multiple disjoint
    - Access time
      - Uniform
      - Hierarchical
  - Physical structure
    - Interconnection
      - Direct (circuit-switched)
      - Logical paths (message-switched)
    - Growth rate
      - Linear
      - Polynomial
    - Concurrency
- Processor-Memory Data Paths
  - Width of Data Path
  - Sharing
    - Simplex
    - Half-duplex
    - Full-duplex
    - Half-multiplexed
    - Full-multiplexed
    - Broadcast
  - Data rate
  - Delay
- I/O system
  - Logical structure
    - I/O initialization
      - Uniform from all processors
      - Partial
    - I/O data transmission
      - Uniform to all processors/memory
      - Partial
    - Access Time
      - Uniform
      - Hierarchical
  - Physical structure
    - Size
    - Data rate
    - Interconnection
      - Direct (circuit-switched)
      - Logic paths (message-switched)
    - Growth rate
      - Linear
      - Polynomial
    - Concurrency
    - Sharing
      - Simplex
      - Half-duplex
      - Full-duplex
      - Half-multiplexed
      - Full-multiplexed
      - Broadcast
- Ratios
  - Memory bandwidth/processor bandwidth
  - I/O bandwidth/memory bandwidth
- Interprocessor communication
  - Interprocessor interrupt
  - Pseudointerrupt device
  - Segment typing
  - Mailboxes

Figure 2.16. Multiple-Processor Design-Space Parameters

2.2 A Methodology for the Design of Multiprocessor Systems

The previous section was devoted to selecting a taxonomy that meets instruction execution requirements. These requirements include the need for a hierarchically organized taxonomy and the need for a taxonomy based on the functional properties of the systems under study. The taxonomy for multiprocessor systems represents a merger of the taxonomies of Siewiorek and of Anderson and Jensen.

This taxonomy will serve as a framework for selecting a system design. It views the multiprocessor design space as a tree structure with design decisions to be made at each internal (nonleaf) node. A design methodology should provide designers with criteria for choosing which branch of the taxonomy tree to take. At any given node in the tree decisions are made based on the trade-offs in reliability, performance and cost. The intersection of these choices is the subspace of feasible designs (see Figure 2.18).

2.2.1 Functional requirements of the system. - Before one begins to design a system, one must understand the top-level functions that the system will perform. This is a crucial activity which requires a serious effort. System functions may be described at several levels--from the topmost level to a description of each software module. Typically, functional descriptions are in informal English prose, but it is also possible to employ formal descriptions; in fact, formal functional specification makes it possible to apply various mathematical models in the verification or validation of the system.

A requirements specification should accompany the functional specification. The requirements specification describes the computational (performance) and reliability required by each of the identified functions.

The aircraft electronics suite cleaves naturally into two components: a sensor-based (or data acquisition) system and a processor-based (or data processing) system. The sensor-based system includes actuating functions as well. The processor-based system performs the computations and arithmetic operations necessary to aircraft electronics. The sensor-based

Dimensions

- Node Types
  - Nonhomogeneous
  - Homogeneous
- Memory System
  - Logical structure of address space
    - Local
    - Shared
      - 1
      - n
      - n(m)
  - Protection
    - None
    - Object
    - Capability
  - Physical structure of memory
    - Size
      - Immediate
      - System
    - Redundancy
      - Replication (*r)
      - Coding
        - Parity (*p)
        - Hamming (*h)
- Memory Switch
  - Logical structure
    - Accessibility
      - All
      - Partial
        - Overlapped
        - Multiple disjoint
    - Access time
      - Uniform
      - Hierarchical
  - Physical structure
    - Interconnection
      - Direct (circuit-switched)
        - Dedicated path
          - Loop
          - Complete interconnection
        - Shared path
          - Central memory
          - Global bus
      - Logical paths (message-switched)
        - Centralized routing
          - Dedicated path
            - Star
            - Loop with central switch
          - Shared path
            - Bus with central switch
        - Decentralized routing
          - Dedicated path
            - Regular network
            - Irregular network
          - Shared path
            - Bus window
    - Growth rate
      - Linear
      - Polynomial
    - Concurrency
    - Redundancy
- Processor-Memory Data Paths
  - Width of Data Path
  - Sharing
    - Simplex
    - Half-duplex
    - Full-duplex
    - Half-multiplexed
    - Full-multiplexed
    - Broadcast
  - Data rate
  - Delay
- I/O system
  - Logical structure
    - I/O initialization
      - Uniform from all processors
      - Partial
    - I/O data transmission
      - Uniform to all processors/memory
      - Partial
    - Access Time
      - Uniform
      - Hierarchical
  - Physical structure
    - Size
    - Data rate
    - Interconnection
      - Direct (circuit-switched)
      - Logic paths (message-switched)
    - Growth rate
      - Linear
      - Polynomial
    - Concurrency
    - Sharing
      - Simplex
      - Half-duplex
      - Full-duplex
      - Half-multiplexed
      - Full-multiplexed
      - Broadcast
    - Redundancy
- Ratios
    - Memory bandwidth/processor bandwidth
    - I/O bandwidth/memory bandwidth
- Interprocessor communication
  - Interprocessor interrupt
  - Pseudointerrupt device
  - Segment typing
  - Mailboxes

Figure 2.17. Multiple-Processor Design-Space Parameters

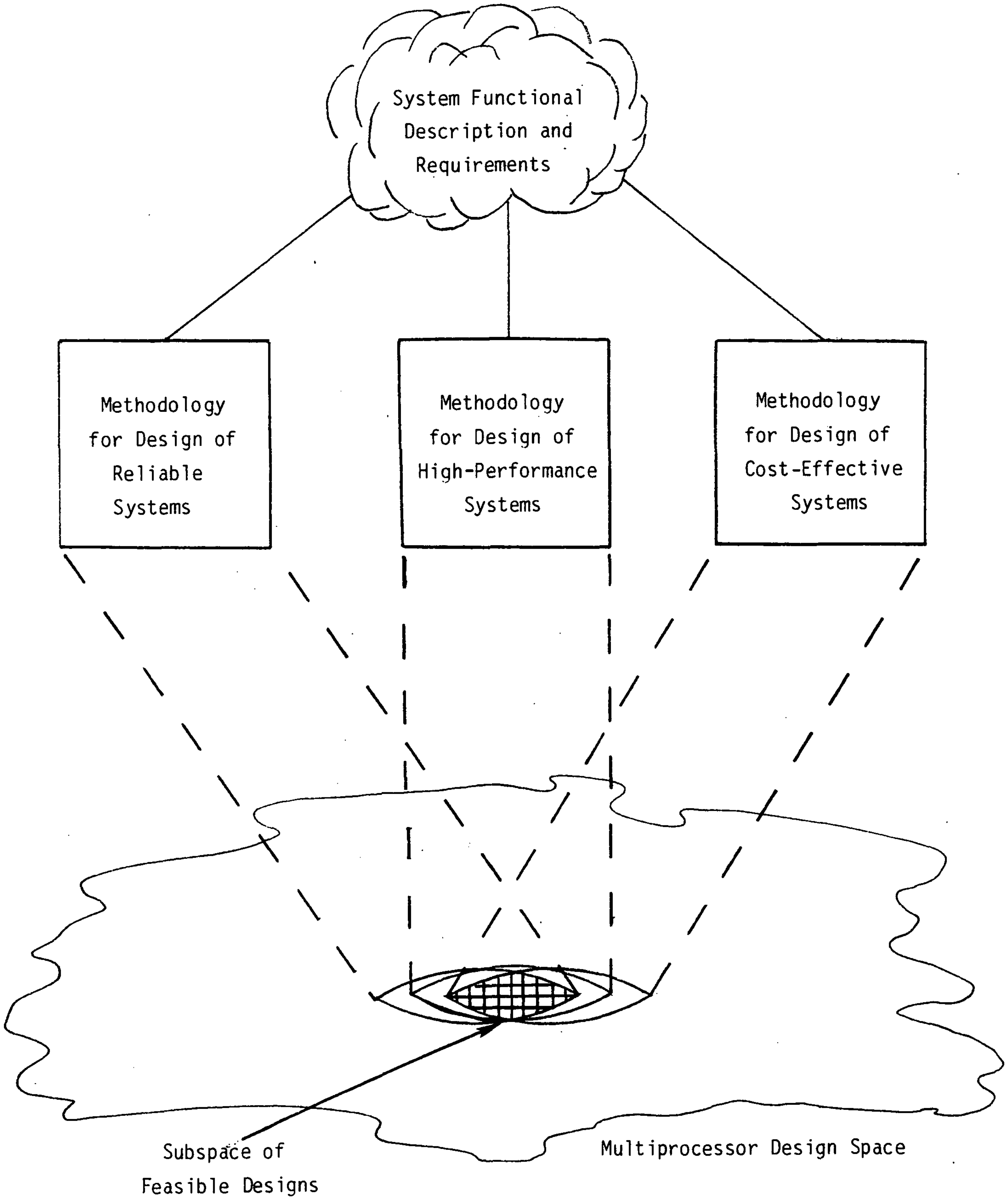

System Functional Description and Requirements
Methodology for Design of Reliable Systems
Methodology for Design of High-Performance Systems
Methodology for Design of Cost-Effective Systems
Subspace of Feasible Designs
Multiprocessor Design Space

Figure 2.18. Elements of the Proposed Design Methodology.

system does signal processing, front-end processing and filtering, as well as system I/O.

Functions to be performed by the processor-based system are shown in Table 2.1. The flight phases in which these functions are required are also tabulated. Tables 2.2 and 2.3 indicate the reliability and computational requirements for various functions. Computational requirements are used to estimate the processor power required to execute the functions; reliability requirements are used to estimate the level of reliability appropriate to the processor-based system. As shown in the tables, aircraft electronic functions have high reliability and computational requirements.

2.2.2 Measures for the evaluation of systems. - In this section a set of measures for evaluating multiprocessor systems are proposed and then used in the design methodology. The significance of measures to the methodology is that they allow system designs to be measured with respect to some desirable characteristic. One can then select those designs whose measures meet the postulated system requirements.

The proposed measures fall into three broad classes: reliability, performance, and cost.

Reliability measures

- failure-effect: the susceptibility of a system to a single failure
- failure-reconfiguration: the ability of a system to operate in a degraded mode once a failure has occurred

Performance measures

- bottleneck: the tendency of a system's throughput to be limited by lowered performance in a subsystem
- communication complexity: the totality of decisions made during communications by source processes, destination processes or switching entities

Table 2.1. Aircraft Electronic Functions for All Flight Phases

Function	Flight Phase						
	Takeoff	Climb Descent	Cruise	Initial Approach	Landing	Missed Approach	Navigational Failure
Attitude Control	-	P	P	P	-	P	P
Flutter Control	-	P	P	-	-	-	-
Load Control	-	S	S	S	P	S	S
Autoland	-	-	-	-	P	-	-
Autopilot	-	P	P	P	B	P	P
Attitude Indicator	P	P	P	P	P	P	P
Inertial Navigation	P	P	P	P	P	P	P
VOR/DME & Multiple DME	-	P	P	P	P	P	-
OMEGA or Satellite	-	B	B	B	B	B	P
Air Data (Navigation)	-	-	-	-	-	-	B
Estimation (Kalman Filter)	-	P	P	P	P	P	B
Flight Data	-	P	P	P	P	P	B
Airspeed, Altitude	P	P	P	P	P	P	P
Graphic Display	-	P	P	P	P	P	-
Text Display	S	S	S	P	P	P	S
Collision Avoidance	P	P	P	P	P	P	P
Data Comm., A/C	P	P	P	P	P	P	P
Data Comm., Air/Ground (DABS)	S	S	S	P	P	P	S
AIDS	S	S	S	S	S	S	S
Instrument Monitor	S	S	S	S	S	S	S
System Monitor	S	S	S	S	S	S	S
Life Support	-	P	P	S	-	-	P
Engine Control	P	P	P	P	P	P	P

Symbols: P = Prime; S = Secondary; B = Backup; - = N/A

Table 2.2. Reliability Requirements

Function	Missed Iterations	Data Protection	Criticality Class*
Attitude Control	2-3	Yes	1
Flutter Control	2-3	No	1
Load Control	2-3	No	3 &/or 5
Autoland	2-3	Yes	1
Autopilot	4-5	Yes	4
Attitude Indicator		No	1
Inertial Navigation	0,4	Yes	2
VOR/DME & Multiple DME	4-5	No	4
OMEGA or Satellite	4-5	No	4
Air Data (Navigation)	4-5	No	4
Estimation (Kalman Filter)	2-3	Yes	4
Flight Data	2-3	No	4
Airspeed, Altitude	2-3	No	4
Graphic Display	2-3	No	4
Text Display	4-5	No	4
Collision Avoidance	1-2	Yes	4
Data Comm., A/C	--	Yes	--
Data Comm., Air/Ground (DABS)	--	No	4
AIDS	4-5	No	5
Instrument Monitor	2-3	Yes	4
System Monitor	2-3	Yes	1-4
Life Support	3-4	Yes	1-4
Engine Control	1-2	Yes	1-2

*Class 1: function immediately critical to safety of flight.
Class 2: function will be critical to safety of flight at a future point in the mission.
Class 3: loss of function requires significant change in mission.
Class 4: loss of function imposes substantial operational penalties.
Class 5: loss of function has undesirable economic consequences.

Table 2.3. Computational Requirements

Function	Rate/Sec.	Period in msec.	Instructions per Iteration	MIPS	Memory (Words)
Attitude Control	20	50	1150	0.023	2075
Flutter Control	250	4	276	0.069	92
Load Control	240	4	58	0.014	60
Autoland	160	6.25	344	0.055	1025
Autopilot	5	200	200	0.001	250
Attitude Indicator	30	33.3	2567	0.077	1310
Inertial Navigation	25	40	1360	0.034	2250
VOR/DME & Multiple DME	5	200	800	0.004	300
OMEGA or Satellite	5	200	800	0.004	505
Air Data (Navigation)	5	200	200	0.001	135
Estimation (Kalman Filter)	0.2	5000	5000	0.001	315
Flight Data	5	200	5600	0.028	550
Airspeed, Altitude	16	62.5	562	0.009	430
Graphic Display	8	125	4000	0.032	6250
Text Display	10	100	1900	0.019	9340
Collision Avoidance	670	1.5	31	0.021	1200
Data Comm., A/C	250	4.0	1200	0.300	610
Data Comm., Air/Ground (DABS)	4	250	250	0.001	562
AIDS	4	250	500	0.002	1300
Instrument Monitor	5	200	2800	0.014	1900
System Monitor	0.5	2000	200	0.001	1000
Life Support	0.5	2000	2000	0.001	1000
Engine Control	33	30	3606	0.119	1500

Cost measures

- cost-modularity: the incremental cost of adding an element
- place-modularity: the degree to which the location or function of an added element is restricted
- connection-flexibility: the ability to employ alternative topologies, given a specific interconnection structure.

These measures are not intended to be quantitative and often take qualitative values (high, moderate, low) when applied to systems.

Table 2.4 shows how different generic families of the taxonomy vary with respect to the identified reliability, performance and cost measures.

2.2.3 Tools for the methodology. - The qualitative measures discussed above are useful in selecting feasible designs. Once the feasible design subspace has been isolated, however, more precision is desired in evaluating the best design for this particular application. Simulation and analytical tools are the time-proven means for the precise evaluation of a given design. Some of these tools are discussed below.

2.2.3.1 Automatic generation of reliability functions for PMS structures: Each of the various multiprocessor systems treated by the taxonomy can be expressed in PMS descriptions. PMS is a well-known hardware description language (See ref. 3), the description of which is beyond the scope of this report. Once a set of feasible systems have been targeted, they may be translated into PMS descriptions.

The motive for describing the systems in PMS notation is the Advanced Interactive Symbolic Evaluation of Reliability (ADVISER) program (See ref. 18). ADVISER is the result of a study to demonstrate the feasibility of building design tools that perform a significant portion of the reliability analysis of systems. ADVISER attempts to relieve the designer of the burden of the traditionally complex, tedious and error-prone reliability analysis of systems. The user of ADVISER essentially provides ADVISER with a PMS description of the system under examination and the failure rate description of various PMS components. ADVISER ultimately produces symbolic reliability functions of the system. The scope of systems analyzable by ADVISER is shown in Figure 2.19.

Table 2.4. Reliability, Performance and Cost of Generic Classes of Multiprocessor Systems.

Measure / System	Reliability Measures						Performance Measures		Cost Measures					
	failure-effect			failure-reconfiguration			bottleneck	communication complexity	cost-modularity			place-modularity		connection flexibility
	PE failure	Path failure	Switch failure	PE failure	Path failure	Switch failure			PE	Path	Switch	PE	Switch	
DDL	poor	poor	-	poor	poor	-	loop bandwidth, PE	low	very good	very good	-	very good	-	-
DDC	good	good	-	good	good	-	none obvious	low	poor	poor	-	good	-	-
DSM	good	poor	-	good	poor	-	memory bandwidth	low	very good to poor	very good	-	very good	-	-
DSB	very good	poor	-	very good	poor	-	bus bandwidth	low to moderate	good	poor	-	good	-	-
ICDS	good to poor	fair	poor	good to poor	fair	poor	switch	moderate	good	poor	-	good	good	poor
ICDL	poor	poor	poor	poor	poor	poor	loop bandwidth, PE, switching	moderate	good	good	-	good	good	fair
ICS	fair	poor	poor	good	poor	poor	bus bandwidth, switching	moderate	good	good	-	good	good	poor
IDDR	good to moder-ate	good to moder-ate	good to moder-ate	poor	poor	poor	none obvious	moderate	poor	poor	poor	poor	poor	poor
IDDI	good	good to fair	good to fair	good	good to fair	good to fair	none obvious	high	good	good	good	good	good	good
IDS	good	fair	fair	good	fair	fair	bus bandwidth, switching	moderate	good	good	good	good	good	good

A program such as ADVISER can then be used to further attenuate the feasible design subspace. The output of ADVISER is simply used to concentrate on those systems with estimated reliability ranges within the window of postulated reliability requirements.

2.2.3.2 Simulation to determine performance: A candidate system should certainly be subject to scrutiny before plans to implement the system are pursued. In the absence of any physical system, simulation becomes a necessity. Simulation is also a very cost-effective means for predicting system behavior and fine-tuning design parameters. Simulation techniques to deal with three performance issues--throughput, deadlock and bottleneck--will be considered below.

The Extendable Computer System Simulator II (ECSS II), a program based on SIMSCRIPT (See ref. 19), is used principally for simulating computer system behavior. It has the capability to model various system components such as processors, memories, I/O and other devices, and queues as well as various attributes of these entities. It can also model certain events related to the entities' functions, e.g., arrival, processing and storage. This tool is publicly available from the Models Directorate of FEDSIM and has the promise of great usefulness in the proposed taxonomy.

Ramamoorthy (See ref. 26) has an approach for predicting maximum performance of a system and for detecting the potential for deadlock in systems. Both approaches are based on concepts from the theory of Petri nets. In modeling systems with Petri nets the problem is that there is no easy way to interpret the system as a Petri net; i.e., this is an art and not a science. If this impediment can be removed, Ramamoorthy's techniques will indeed be attractive.

Han's (See ref. 16) approach to identifying system bottlenecks is also based on Petri nets. The same considerations that apply to Ramamoorthy's work also apply here. If the details of Petri net modeling can be worked out, then bottleneck identification will be feasible.

2.3 A Strawman Fault-Tolerant Integrated Aircraft Electronic System

For the purposes of illustration and discussion a strawman system is presented below. Examining Table 2.4 gives a rough estimate of the total

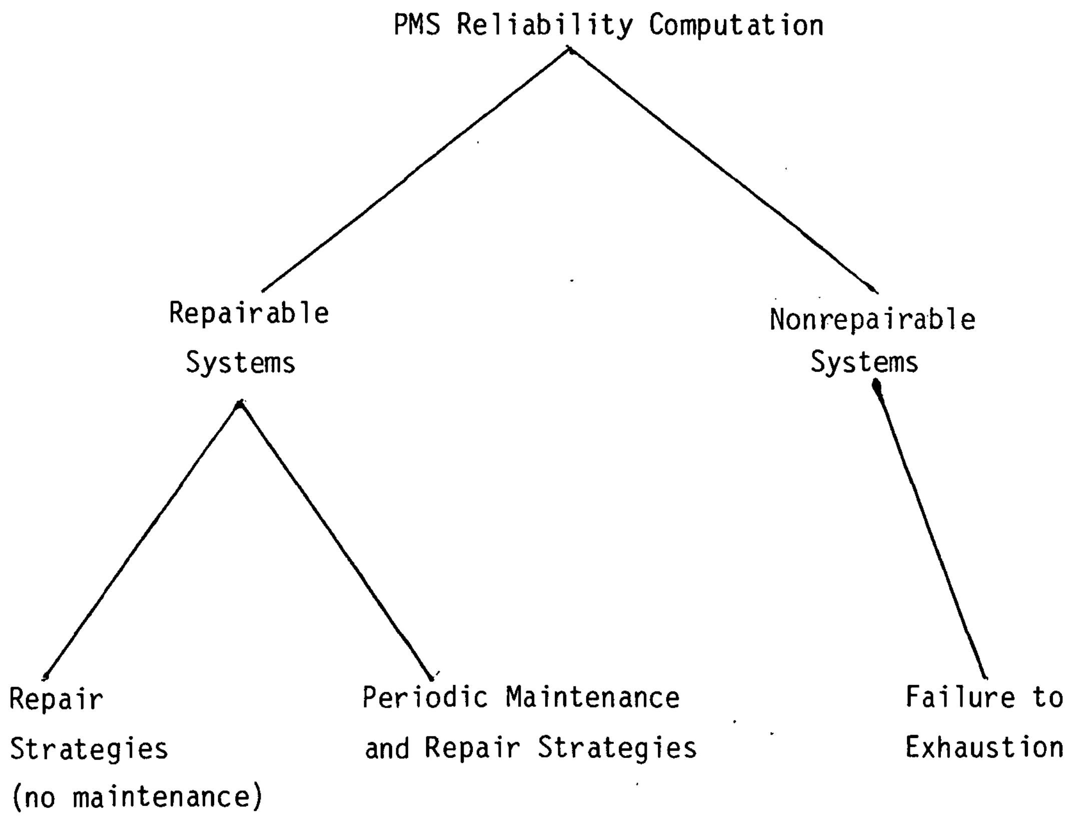

PMS Reliability Computation
Repairable Systems
Nonrepairable Systems
Repair Strategies (no maintenance)
Periodic Maintenance and Repair Strategies
Failure to Exhaustion

Figure 2.19. Reliability modeling at the PMS level.

figure of merit for each of the 10 different interconnection architectures. If reliability, performance and cost are weighted as three, two and one, respectively, then a ranking of the different interconnections may be obtained. In calculating the figure of merit, "good" is interpreted as three, "fair" is interpreted as two and "poor" is interpreted as one. The relative ranking is as follows.

1) DDC
2) DSM
3) IDDI
4) DSB
5) ICS
6) IDS
7) ICDS
8) DDL
9) IDDR
10) ICDL

The use of the DDC architecture, the highest ranking, will be considered in a fault-tolerant integrated aircraft electronic system. The system design will be divided into the design of the sensor-actuator subsystem and the processor system. A complete interconnection structure can be immediately eliminated from the sensor-actuator subsystem, since information exchange between sensors and actuators will be limited to a few subsets of the entire subsystem. However, complete interconnection of sensor-actuators with processing elements will be included. Furthermore, all processors will be completely interconnected and fully replicated for the purposes of fault-tolerance and reconfiguration. Processors will also have complete interfacing with the sensors and actuators. The strawman system is depicted in Figure 2.20.

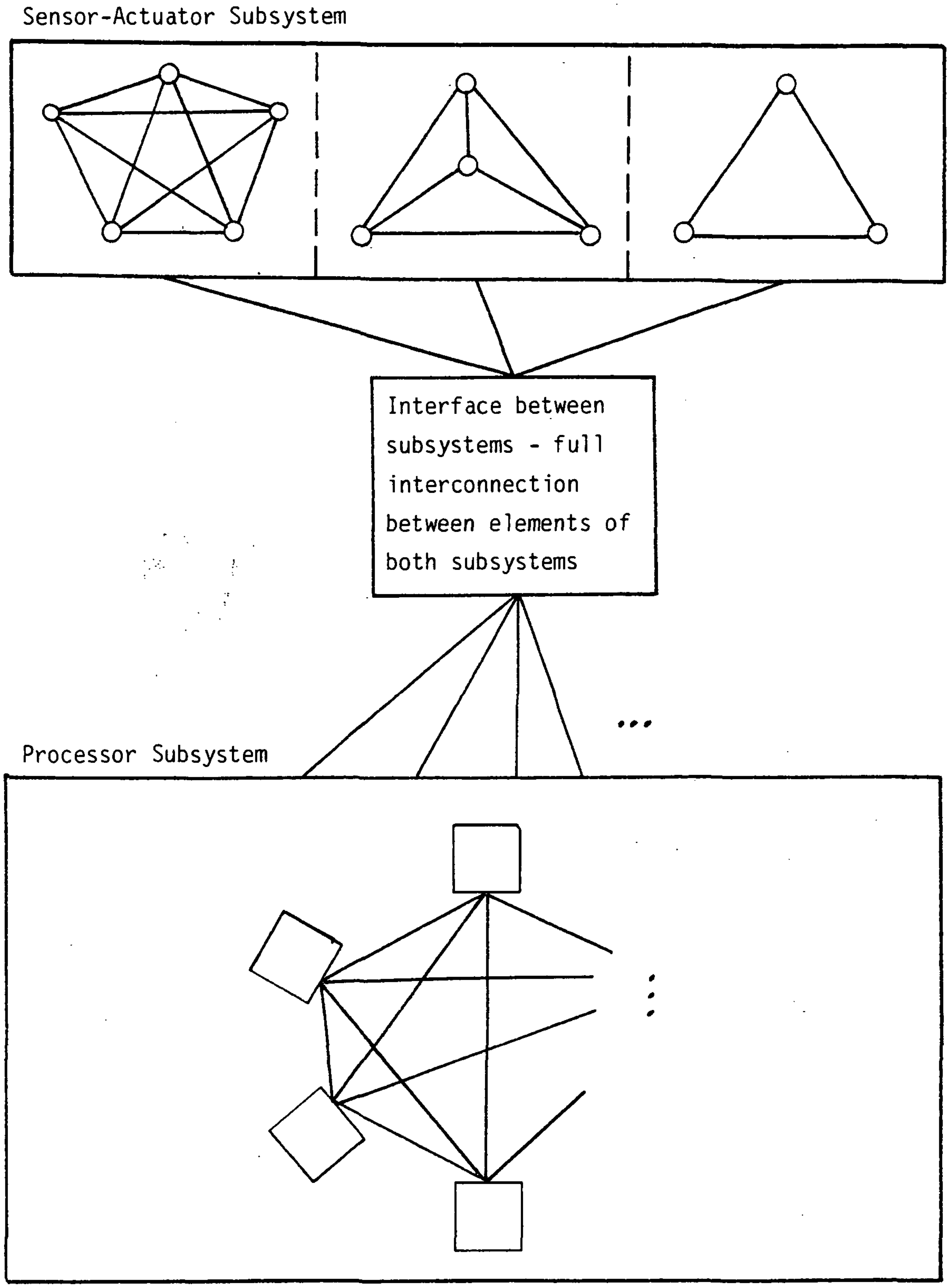

Sensor-Actuator Subsystem
Interface between subsystems - full interconnection between elements of both subsystems
Processor Subsystem

Figure 2.20. DDC Architecture for Aircraft Electronics.

3.0 ALGORITHMS AND CONTROL STRUCTURES FOR AN INTEGRATED AIRCRAFT ELECTRONIC SYSTEM

In the previous chapter, the design of the hardware for an integrated aircraft electronic system was discussed. In this chapter, the software aspects of system design will be considered, first in terms of the management of system resources and then in terms of the role of software design in the overall system design.

3.1 Managing System Resources

3.1.1 Scheduling of real-time fault-tolerant systems. - In this section the general background material that is necessary for understanding how systems are scheduled and how scheduling methods are evaluated is provided, followed by the specific problem of scheduling tasks in aircraft electronic systems.

3.1.1.1 General scheduling theory: Scheduling is a crucial aspect of the problem of efficiently managing the resources of a computing system. If a system has available a finite amount of certain resources (e.g. processors, devices, memory) to be used by a group of competing consumers (generally speaking, tasks or jobs), then it is imperative to schedule these consumers. The act of scheduling the consumers is essentially a directive which notifies the system when and for how long a consumer shall use specific resources.

Scheduling is necessary from the point of view of making the system work: the tasks to be performed must be scheduled and dispatched (started) in some specific order and at certain points in time if the system is to even begin performing its function. Hence, a mechanism for producing and initiating schedules is inherently present in all operational systems.

The real payoff, however, stems from the fact that different schedules may have different overall system implications. One major system implication is the cost of executing a set of tasks. In general, two different schedules will have two different costs. Clearly, one wishes to choose the schedule with the smaller cost. Another important system implication involves the relative importance or criticality of the tasks. One would like to be confident that the most critical tasks will always have enough resources available to fully perform their functions. This could entail preempting tasks of lesser criticality, but the system may nevertheless function acceptably even when the least critical tasks are offered less than their requested complement of resources. A schedule controls the tasks' access to resources; therefore, scheduling plays a key role in the effort to ensure that critical tasks are provided the resources they require (possibly at the expense of less critical tasks).

The choice of the "right" scheduling strategy will thus yield a system with desirable characteristics: reduced running costs, assurance that critical tasks will be performed, and efficient use of resources.

In a computing system dedicated to guidance and flight controls, tasks will have to be repeatedly executed in a real-time manner. More specifically, the existence of some number of processing elements can be assumed. The number of processors may decrease as the result of hardware failures. The tasks will be ordered according to their relative importance to the given flight phase. These are the essential elements of the system that is to be scheduled.

Generally in scheduling multiprocessor systems, one begins with a set of m (identical) processors

$$P = \{P_1, P_2, \ldots, P_m\}.$$

A <u>task system</u> for a set of processors P is a triple $(T, <, \mu)$ where:

1) $T = \{T_1, T_2, \ldots, T_n\}$ is a set of tasks to be executed.
2) $<$ is an irreflexive partial ordering on T indicating the precedence constraints of the tasks; $T_i < T_j$ means that T_i must be completely executed before T_j can be started.
3) μ: $T \longrightarrow \{x:\ x \text{ is a positive real number}\}$ is a function specifying the execution time required by each task.

Note that a task has the same execution time for each processor.

If one would like to schedule the tasks on various processors, given such a task system, a list is created that tells, for each task, the time at which it is to be started and the processor on which it is to run. This schedule must be consistent: two tasks cannot be scheduled to run on the same processor at any given time. These notions are formalized by defining a schedule for a task system $(T, <, \mu)$ on processor set P to be a function

$$S: \{x: \text{ where } x \text{ is a non-negative real number}\} \longrightarrow 2^T$$

(2^T is the set of all subsets of T). The function S must satisfy the following properties:

1) $|S(t)| \leq m$ for all $t \geq 0$. No more than m processors are running at any time.
2) There is a minimal real number $\omega(S)$ such that for all $t \geq \omega(S)$ $S(t) = \emptyset$. $\omega(S)$ is the total execution time for S.
3) $\bigcup_{0 \leq t \leq \omega(S)} S(t) = T$. All tasks get scheduled.
4) Let $t_s(T_i) = \inf_{T_i \in S(t)} t$ be the starting time of task T_i.
   $T_i \in S(t)$ if and only if
   $t_s(T_i) \leq t \leq t_s(T_i) + \mu(T_i)$.
   A task runs for only one uninterrupted time segment.
5) If $T_i < T_j$, then $t_s(t_i) + \mu(T_i) \leq t_s(T_j)$. A task may not be started until its predecessors are completed.

Scheduling a task system on multiprocessors to minimize the schedule's total execution time ($\omega(S)$, or simply ω if no possibility of confusion exists) is thought to be an inherently difficult problem (i.e., the problem of determining an optimal schedule is NP-hard). This is highly unfortunate, as schedulers are generally required to make decisions in real-time. One is therefore forced to consider scheduling algorithms that do not give optimal schedules. These nonoptimal algorithms are sometimes referred to as "heuristic" algorithms. Heuristics are, strictly speaking, not algorithms but rather loosely applied approaches to the solution of a problem (i.e., a "rule of thumb"); nonetheless, the oxymoronic phrase "heuristic algorithm" will often be employed to refer to an algorithm that does not in all cases produce an optimal solution.

If one accepts the thesis that the general multiprocessor scheduling problem is intractable, then one must resolve to live with lowered expectations. There are basically two ways to deal with intractability. These are:

1) Design efficient algorithms that produce suboptimal solutions.
2) Restrict the given problem so that one can design efficient algorithms that still produce optimal solutions.

Alternative (1) is essentially an attempt to discover algorithms whose executions require reasonable amounts of time and space and whose outputs are suboptimal, yet acceptable. Alternative (2) attempts to discover nontrivial, useful subproblems of the original problem for which algorithms with reasonable time and space requirements can be designed. Furthermore, the restricted algorithm should produce optimal results.

It is necessary to digress momentarily in order to clarify what constitutes a reasonable algorithm or an acceptable solution. In the jargon of computer science a reasonable algorithm is one that requires space and time that grow no faster than a polynomial function of the problem size. Even this is unrealistic since polynomials may grow very rapidly (e.g., $10^{15} \bullet x^{100}$). In practice it is common to expect a reasonable algorithm to run in quintic time and space. The other subtlety involves deciding when an algorithm produces a "good" suboptimal solution. Usually, given an algorithm that is suboptimal, the question is posed: "Is there a guarantee that this algorithm will never give a result that is 'x' percent worse than optimal?" Given a heuristic algorithm HA for a problem P one can speak of the <u>absolute performance ratio</u> for an instance I of the problem. This is defined to be $HA(I)/OPT_P(I)$, where $OPT_P(I)$ is the optimum for instance I of problem P. In general, one hopes to find a constant upper (lower) bound on the absolute performance ratio for minimization (maximization) problems. It is desirable in practice to establish an upper bound of no more than 2 or 3 (100 - 200% more than the optimum). A less desirable, but more common, situation is when the upper bound is a function of the problem parameters rather than a constant.

In the face of intractability the two alternatives discussed above have actually been employed by computer scientists.

As an example of alternative (2), one could use a very simple algorithm to schedule a task system on a set of processors. One such simple algorithm is the priority list scheduling algorithm given in Figure 3.1. A classical result of Graham (See ref. 14) is the following.

<u>Theorem 3.1</u>: Let ω_0 be the smallest possible total execution time for a task system $(T, <, \mu)$ on a set of m identical processors P and ω_{PL} be the total execution time for $(T, <, \mu)$ on P when scheduled by a priority list discipline. Then

$$\frac{\omega_{PL}}{\omega_0} \leq 2 - \frac{1}{m} .$$

Furthermore, this bound is tight; i.e., an infinite number of task systems $(T_i, <, \mu)$ exist such that

$$\frac{\omega_{PL}}{\omega_0} = 2 - \frac{1}{m},$$

where ω_{PL} is the total execution time of $(T_i, <, \mu)$ scheduled on P by a priority list discipline.

The above theorem is a guarantee that the priority list scheduling algorithm of Figure 3.1 will always produce a schedule that is completed in less than twice the optimal completion time. Note also that the algorithm is a polynomial time algorithm (running in time proportional to mn). This is a premier example of a fast algorithm that gives reasonably good (but suboptimal) schedules.

3.1.1.2 Scheduling repeatedly requested tasks subject to hard real-time deadlines: The scheduling problems discussed in the previous section seek to schedule tasks that are executed only once and are not subject to any deadlines. In a real-time environment, especially in control systems such as flight controls and guidance, the tasks will typically be repeatedly requested and executed. Each task will have to be completed before its next request--this constitutes the task's inherent deadline.

The implicit assumptions in this application are:

Input

A set of n independent tasks with execution times mu[1..n] to be scheduled on m processors.

Output

A schedule specified as

assign[1..n] - task i is assigned to processor assign[i]
start[1..n] - the task's start time
finish[1..n] - the task's finish time

Algorithm

```
begin
    omega ← 0;
    for j ← 1 to m do free[j] ← 0 od;
    for i ← 1 to n
       do
          min ← 1;
          for j ← 1 to m
              do if free[min] > free[j] then  min ← j fi  od;
          assign[i] ← min;
          start[i] ← free[min];
          finish[i] ← free[min] + mu[i];
          free[min] ← finish[i];
          if omega < finish[i] then omega ← finish[i] fi
       od
end.
```

Figure 3.1. Priority List Scheduling Algorithm

1) The requests for each task consist of an infinite number of equally spaced requests.

2) Each task must be completed before its next request.

3) The tasks are independent of each other.

4) Each task has a constant run-time.

5) The execution of a task can be divided into discrete chunks, i.e., tasks can be scheduled preemptively.

First, in terms of how to schedule such a set of tasks on a single processor, the fundamental question is: "How does one know when a set of tasks can be scheduled on a processor?" This question was answered by Liu and Layland (See ref. 22). If the utilization of a task is defined as its run-time divided by its request interval, then a set of tasks may be scheduled on a processor, provided that the sum of all task utilizations does not exceed certain scheduling constants. The scheduling constants are related to the particular scheduling algorithms used. Liu and Layland studied two algorithms. In the rate monotonic priority algorithm the highest priorities are assigned to tasks with the highest utilizations; tasks are then preemptively scheduled by using a standard priority algorithm. In the deadline-driven algorithm, each task's priority at any given time is determined by the proximity of its deadline; tasks are continuously being reprioritized and rescheduled (preemptively). Thus, according to Liu and Layland, a set of tasks may be preemptively scheduled on a processor by either of the above algorithms as long as certain constraints are satisfied by the set of tasks. The constraint is that the sum of the tasks' utilizations shall not exceed certain scheduling constraints. For the rate monotonic priority algorithm and the deadline-driven algorithm, the scheduling constants in this application are ln 2 (approximately .69) and 1, respectively.

Dhall and Liu (See ref. 10) studied a generalization of the problem above where multiprocessors are available. Their solution reduces the multiprocessor problem to first allocating subsets of tasks to the processors and then individually applying either the rate-monotonic priority algorithm or the deadline-driven algorithm individually to each processor and the tasks allocated to it. The problem of scheduling real-time fault-tolerant systems then becomes one of allocating a set of replicated tasks to a collection of processors.

3.1.2 Task allocation for the derivation of timely schedules. - Several classes of problems play key roles in the integration of fault-tolerant guidance and flight control systems. In the past, these problems were not addressed in any systematic manner; system designers relied on intuition and ad hoc methods in solving these problems, or the problems were completely ignored.

Some specific classes of integration problems that warrant systematic approaches are:

- efficient resource allocation in fault-tolerant multiprocessor systems
- reliable restructuring of control laws in the face of altered flight requirements
- use of parallelism in conventional control theory
- methods for simplifying the programming and use of fault-tolerant multiprocessor systems
- techniques for improving software reliability in fault-tolerant multiprocessor systems

One very important topic in guidance and flight control systems is related to two of these classes. This topic--the sizing, allocating, and scheduling of a system of processor-memories and tasks--involves the problem classes, efficient resource allocation and simplification of programming.

Sizing, resource allocation and scheduling may be approached in two ways: statically or dynamically. The static discipline normally requires off-line precompilation of the allocation and schedule, relying heavily on tabular methods, thus requiring some form of involvement on the part of the programmer for table creation. The dynamic discipline frees the programmer from this onus and requires less memory.

An interesting problem addressed in the design of SIFT (See ref. 33) is the allocation problem, which requires that tasks and processor-memories be mutually allocated in such a way that processor-memory load is balanced. A special-purpose technique has been employed (See ref. 33) and the results appear to be satisfactory. The results were obtained for SIFT with a small, hypothesized task set for which strong assumptions about the processor and task characteristics were made. There is no indication that the allocation derived for SIFT could not be improved. The design of large fault-tolerant multiprocessor systems with static allocation will require that this problem be solved in a generic way. The initial objective is to provide a general-purpose solution to the problem which will optimize the allocation with respect to selected target functions.

3.1.2.1 Mathematical formulation of the task allocation problem: In considering a problem in which a set of tasks and a collection of processors are given, the most general case is one in which all processors may be different, although a uniform processor set is most likely to be encountered in fault-tolerant systems. The characteristics of each processor, viz., speed (in instructions per second) and memory capacity (in words or bytes) are assumed to be known.

Information is also available about the tasks, which are assumed to be periodically executed. The information available on the tasks includes iteration rates (the number of times the task is requested per second), the number of instructions executed per iteration, and the amount of memory required by the tasks (i.e., the space occupied by the program plus the maximum size achieved by its activation records). Other information includes the active and passive replication factors of the tasks: an actively replicated task is executed on each processor to which it is allocated, whereas a passively replicated task is not executed under normal conditions. The passively replicated task resides in the memory of a processor and is executed only in the event that one of its active instances is unable to be executed. In that case, the passive task becomes active and the task system is reconfigured by allowing the formerly passive task to execute on its assigned processor. Clearly, passive allocation is employed for the most critical tasks.

It is well known (See ref. 22) that periodic tasks may be well scheduled on a processor as long as the tasks' collective utilization of the processor is maintained below a certain threshold. For example, a set of periodic tasks may be scheduled on a processor using the rate-monotonic priority algorithm or the deadline-driven algorithm, if the utilization of the processor is ln 2 or 1, respectively.

The problem is essentially to assign the tasks to processors so that no single processor is utilized much more than any other. For example, if a two-processor system is considered and a set of five tasks are assigned, the tasks are T_1, T_2, T_3, T_4, T_5 and the processors are P_1 and P_2. The tasks have respective utilization factors of .5, .4, .3, .2, .2. If each task is to be actively replicated only once and no task is to be passively replicated, there are 32 different assignments possible. Assigning T_1, T_3 to P and T_2, T_4, T_5 to P_2 achieves the optimum since both processors obtain a utilization of .8, indicating that perfect balance in utilization has been achieved.

In the allocation problem one is given m (identical) processors and n tasks are to be actively and passively allocated to distinct processors. One has available certain information about the processors (speed, memory capacity, etc.) and the tasks (executable instructions, memory requirements, replication factor, etc.). The tasks are to be assigned to the processors so as to achieve a balance of processor utilization and memory consumption. This problem is a formalization of a problem encountered in the design of SIFT system (See ref. 32,33).

One way to effect this assignment is by solving the following non-linear (quadratic) integer (0-1) program:

minimize

$$\delta + \epsilon$$

subject to

$$\sum_{1 \le j \le m} x_{ij} = a_i \quad (1 \le i \le n)$$

$$\sum_{1 \le j \le m} y_{ij} = p_i \quad (1 \le i \le n)$$

$$x_{ij} + y_{ij} \le 1 \quad (1 \le i \le n,\ 1 \le j \le m)$$

$$\sum_{1\le i\le n} x_{ij}U_{ij} \le f_j \quad (1\le j\le m)$$

$$\sum_{1\le i\le n} (x_{ij} + y_{ij})M_i \le C_j \quad (1\le j\le m)$$

$$x_{ij},\ y_{ij} \in \{0,1\} \quad (1\le i\le n,\ 1\le j\le m)$$

The symbols are explained:

$U_{ij} = I_i/T_iR_j \quad (1\le i\le n,\ 1\le j\le m)$ task i's utilization of processor j

$V_j = \sum_{1\le i\le n} (x_{ij} + y_{ij})M_i \quad (1\le j\le m)$ processor j's memory loading

$\delta = \sum_{1\le j\le m} \left(\sum_{1\le i\le n} x_{ij}U_{ij} - \sum_{1\le l\le m} \sum_{1\le k\le n} x_{kl}U_{kl}/m\right)^2$ variance in processor utilization

$\epsilon = \sum_{1\le j\le m} \left(V_j - \sum_{1\le k\le m} V_k/m\right)^2$ variance in processor memory loading

$x_{ij} \quad (1\le i\le n,\ 1\le j\le m) = \begin{cases} 1 & \text{if task i is actively assigned to processor j} \\ 0 & \text{otherwise} \end{cases}$

$y_{ij} \quad (1\le i\le n,\ 1\le j\le m) = \begin{cases} 1 & \text{if task i is passively assigned to processor j} \\ 0 & \text{otherwise} \end{cases}$

$a_i \quad (1\le i\le n)$ = number of times task i is to be actively replicated

$p_i \quad (1\le i\le n)$ = number of times task i is to be passively replicated

$I_i \quad (1\le i\le n)$ = number of instructions to be executed by task i per iteration

T_i	$(1 \leq i \leq n)$	=	iteration period for task i
R_j	$(1 \leq j \leq m)$	=	speed (in instructions per second) of processor j
f_j	$(1 \leq j \leq m)$	=	scheduling constant for processor j (e.g., ln 2)
M_i	$(1 \leq i \leq n)$	=	memory required by task i
C_j	$(1 \leq j \leq m)$	=	memory capacity of processor j

There are many possible reasons to achieve a balanced loading of the processors and memories:

1) The effective failure rate of a processor may be related to its load (utilization). In a system with otherwise homogeneous processors one will want to equalize their effective failure rates.

2) The time to reconfigure the tasks assigned to a processing unit upon its failure will certainly depend upon its loading. Since which unit will fail first is not known ahead of time, the reconfiguration time is minimized by equal (or nearly equal) loading.

3) Since, in general, many tasks are assigned to a processor, queueing for its service will take place. It is a well known result of queueing theory that the startup delay for tasks ("waiting time") is a highly nonlinear function of server utilization. Clearly, one would not want any one processor heavily utilized, thus creating nonacceptable startup delays for tasks assigned to it.

For convenience, the problem discussed above will be referred to as the processor utilization and memory balancing problem (PUMBP). PUMBP requires the optimization of a nonlinear objective function with decision variables that take on zero-one values and are subject to linear constraints. In practice, PUMBP may take up to 200 decision variables and 150 constraining inequalities. There is a strong reason to believe that such an instance of PUMBP would defy solution by traditional methods (e.g., implicit enumeration, branch and bound). For instance, all known algorithms for the quadratic assignment problem blow up with about a dozen decision variables.

One alternative is to try to reduce this computational burden. To this end, a variant of the PUMBP will be considered.

This new problem is motivated by the fact that in today's technology, memory is plentiful and inexpensive. In contrast, processor time is at a premium. Therefore, in the new problem those decisions affecting memory loading will be ignored and processor utilization will be the primary concern. The new problem, which will be referred to as the processor utilization balancing problem (PUBP), has the following formulation:

minimize δ

subject to constraints

$$\sum_{1\le j\le m} x_{ij} = a_i \qquad (1\le i\le n)$$

$$\sum_{1\le i\le n} x_{ij}U_{ij} \le f \qquad (1\le j\le m)$$

$$x_{ij} \in \{0,1\} \qquad (1\le i\le n,\ 1\le j\le m)$$

where all variables are defined as above.

This formulation represents a slight improvement over PUMBP: the number of decision variables is halved, and the number of constraints is drastically reduced. One is still, however, minimizing a nonlinear objective function. PUBP can be completely linearized, thus reducing the problem to a standard integer (0-1) linear program. The transformed PUBP is as follows:

$$\text{minimize} \sum_{2\le j\le m}\Big(\sum_{1\le i\le n} x_{ij-1}U_{ij-1} - \sum_{1\le k\le n} x_{kj}U_{kj}\Big).$$

subject to the constraints

$$\sum_{1\le j\le m} x_{ij} = a_i \quad (1\le i\le n)$$

$$\sum_{1\le i\le n} x_{i1}U_{i1} \le f$$

$$\sum_{1\le i\le n} x_{ij-1}U_{ij-1} \ge \sum_{1\le k\le n} x_{kj}U_{kj} \quad (2\le j\le m)$$

$$x_{ij} \in \{0,1\} \quad (1\le i\le n,\ 1\le j\le m)$$

Observe that the objective function simplifies to

$$\text{minimize} \sum_{1\le i\le n} x_{i1}U_{i1} - \sum_{1\le k\le n} x_{km}U_{km}$$

since the intermediate values of j mutually cancel each other. Thus, in this second problem an attempt was made to minimize the (statistical) range of the processors' utilization factors, whereas in the first problem an attempt was made to minimize the variance of the processors' utilization factors. Both range and variance are measures of statistical dispersion, although variance is generally considered the superior measure. With stronger conditions on the problem structure, however, it may be possible to find an assignment that minimizes variance by minimizing range, the latter procedure being easier to perform.

Using well-known solution techniques for zero-one programming, it may be possible to efficiently solve the above class of problems (See ref. 5,20,28,31).

3.1.2.2 Heuristic approach to the task allocation problem: In their pure forms PUMBP and PUBP seek to find optimal solutions. Solving these problems to optimality may not be possible in all cases without expending significant processing effort since both problems are NP-hard. One should, therefore, consider efficient algorithms for PUMBP and PUBP that produce good (that is, feasible and close to optimum)--though not necessarily optimal--solutions.

Examples of such heuristics are discussed below. As explained previously, there are n tasks, with each task replicated some number of times on m processors. Replicated tasks are to reside on distinct processors.

One attempt at a heuristic solution is to make a tableau of task instances.

$$\begin{array}{cccc} t_{11} & t_{12} & \cdots & t_{1a_1} \\ t_{21} & t_{22} & \cdots & t_{2a_2} \\ & \cdots & & \\ t_{n1} & t_{n2} & \cdots & t_{na_n} \end{array}$$

Here t_{ij} represents the jth instance of the ith task. Next, tasks are allocated by columns; i.e., t_{11}, t_{21}, ... t_{n1} are allocated to however many of the m processors they require. This is done by the following algorithm and has the advantage of allocating different instances of a task to different processors.

Suppose one wishes to allocate n tasks with utilization factors U_1, U_2, ..., U_n to m processors. Assume $U_1 \geq U_2 \geq \ldots U_n$. An algorithm to do this allocation is as follows:

```
for  i <-- 1  to  n  do
   UMin <-- U[1];
   jMin <-- 1;
   for  j <-- 2  to  m  do
      if  UMin > U[j]  then
         UMin <-- U[j];
         jMin <-- j  fi
   od;
     PU[jMin] <-- PU[jMin] + U[i]
od;
```

In the following discussion, this and other approximation algorithms (See ref. 7,9,21) and the bounds on their behavior with respect to the optimal algorithm are explored. The relationship, if any, with other similar efforts (See ref. 5,15,27) is also pursued.

3.1.2.3 Evaluation of the heuristic algorithm for task allocation*: The problem was to assign replicated tasks to different processors to achieve a balance in the processor utilization, while ensuring a separation of tasks. A computer program was written to implement a heuristic algorithm (see Appendix A). The program was run using values 4, 5, and 6 for the number of processors (computer output enclosed in Appendix C). Table 3.1 shows the utilization of alternative experiments. It is apparent from the data in Table 3.1 that in each case a perfect balance of processor utilization is achieved.

So far the concept of memory utilization has been completely ignored. A check on memory utilization with five processors revealed that in three out of five cases memory exceeds the assumed memory capacity of 20 kilowords per processor. A perusal of data (See Table V-3 in ref. 33) confirmed that five processors are not enough for assignment of all tasks. Note that the allocation of task in Table V-4 (See ref. 33) is not feasible. The memory requirement of processor 1 is 26,495, exceeding the capacity of 20,000.

Let USUM = total utilization requirement of all tasks,
MSUM = total memory requirement of all tasks,
IR = number of replications per tasks, and
NP = No. of processors required.

Also note that
utilization capacity of a processor = .5
memory capacity of a processor = 20,000

Then

$$NP = \text{Max}\left(\frac{2*(USUM*IR)}{.5}\right), \left(\frac{(MSUM*IR)}{20{,}000} + 1\right)$$

The program was modified to determine the number of processors by making use of the above expression for NP (see Appendix B). The results show a perfect balance in processor utilization with six processors, but

*Section 3.1.2.3 was contributed by Veena Adlakha, North Carolina Central University.

again memory exceeds the processor capacity of 20 kilowords in three cases (see Appendix C).

The program was further modified to place a check on memory at each step. To reiterate, the program arranges the processors in ascending order in terms of accumulated processor utilization. Before going down the list and assigning replications of a task, the program checks the total memory, if the current task is assigned to that processor. If the total memory exceeds the memory capacity of 20,000, the possiblity of assigning the task to the next processor is explored. Table 3.2 presents the results of this experiment (see Appendix C). It is clearly established that a balance in both processor memory and processor utilization is achieved.

3.1.2.4 Performance guarantees for a restricted task allocation problem: In this case, the following problem is considered. There are m processors and n tasks. All processors are identical, and all tasks have known utilizations u_i $(1 \leq i \leq n)$. One desires to assign each task to two separate processors in such a way that the statistical variance of the processors' utilization is minimized. (The utilization of a processor is defined to be the sum of the utilizations of all tasks assigned to the processor.) This subproblem of PUBP will be referred to as the BALANCE problem.

The problem described above arises in connection with a real-life problem in the design of fault-tolerant real-time computing systems. The scenario is a fault-tolerant system in which two copies of a process are concurrently executing on different processors. Thus a single faulty processor may be identified by voting on the processors' results.*

The problem can be made mathematically precise as follows:

$$\text{Minimize} \quad \frac{1}{m}\sum_j \Big(\sum_i x_{ij}u_i\Big)^2 - \Big(\frac{1}{m}\sum_j \sum_i x_{ij}u_i\Big)^2$$

subject to the constraints

$$\sum_j x_{ij} = 2 \qquad (1 \leq i \leq n)$$

$$x_{ij} \in \{0,1\} \qquad (1 \leq i \leq n,\ 1 \leq j \leq m)$$

* The results presented are valid for processes with arbitrary replication factors, but the proofs are more complicated.

Table 3.1

Processor Utilization

	Number of Processors		
Processor	4	5	6
1	.420	.321	.268
2	.420	.322	.268
3	.420	.321	.268
4	.420	.322	.268
5	--	.322	.268
6	--	--	.268

Table 3.2

Processor Utilization and Memory

Number of Processors = 6

Processor	Utilization	Memory
1	.268	16532
2	.268	16532
3	.268	16532
4	.268	17427
5	.268	17427
6	.268	17427
Total	1.608	101877

where

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to processor } j \\ 0 & \text{otherwise} \end{cases} \qquad (1 \le i \le n,\ 1 \le j \le m)$$

u_i = task i's utilization $(1 \le i \le n)$

One can now show that BALANCE is NP-hard.

3.1.2.4.1 Difficulty of the restricted task allocation problem.

Theorem 3.2: BALANCE is NP-hard.

Proof: NP-hardness can be shown by reduction to PARTITION.

The decision problem PARTITION is formulated as follows (See ref. 13)

Instance: Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

Question: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$?

BALANCE is reformulated as a decision problem.

Instance: Finite set A, a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, positive integer m and positive real number V.

Question: Can A be partitioned into m subsets $A_1, \ldots, A_m$ such that for each $a \in A$ there are exactly two subsets A_i, A_j with $a \in A_i$ and $a \in A_j$ and

$$\frac{1}{m}\sum_{j=1}^{m}\left(\sum_{a \in A_j} s(a)\right)^2 - \left(\sum_{a \in A} 2s(a)/m\right)^2 \le V?$$

One can now show that any instance of PARTITION can be polynomially transformed into an equivalent instance of BALANCE. Given any instance I of PARTITION consisting of finite set A and a size $s(a) \in \mathbb{Z}^+$ for each

$a \in A$, one can solve the instance I' of BALANCE consisting of the same set A, the same size function s, m = 4 and V = 0.

<u>Claim</u>: The answer to the instance I of PARTITION is affirmative if and only if the answer to the instance I' of BALANCE is affirmative.

<u>Proof of Claim</u>: It is clear that if the answer to I is affirmative then the answer to I' is affirmative as well. One simply takes $A_1 = A_2 = A'$ and $A_3 = A_4 = A-A'$.

If the answer to I' is affirmative, then

$s(A_1) = s(A_2) = s(A_3) = s(A_4)$. Let $B_1 = A_1 \cap A_2$ and $B_2 = A_3 \cap A_4$.

A_1	B_1	C_1
A_2	B_1	C_2
A_3	B_2	C_3
A_4	B_2	C_4

Let $C_1 = A_1 - B_1$, $C_2 = A_2 - B_1$, $C_3 = A_3 - B_2$, $C_4 = A_4 - B_2$.

Then, $C_1 \cup C_2 = C_3 \cup C_4$.

Hence, $\sum_{a \in B_1} s(a) = \sum_{a \in B_2} s(a)$.

Therefore, $\sum_{a \in C_i} s(a) = \sum_{a \in C_j} s(a)$ for $1 \leq i, j \leq 4$.

This implies that A may be partitioned as $A_1' = B_2 \cup C_2$, $A_2' = B_1 \cup C_1$, $A_3' = B_2 \cup C_2$, $A_4' = B_2 \cup C_2$, and each a A is in exactly two of the A_i'.

Moreover,

$$\sum_{a \in A_i'} s(a) = \sum_{a \in A_j'} s(a) \quad \text{for } 1 \leq i, j \leq 4.$$

Taking $A' = A_1'$ and $A - A' = A_3'$, the answer to instance I of PARTITION is affirmative.

3.1.2.4.2. An approximation algorithm and its performance guarantee.

The balance problem is NP-hard. Thus one should to consider approximation algorithms with efficient execution times. The following algorithm is one example. It will be referred to as algorithm "A".

```
0)  i <-- 1.
1)  Find the two least utilized processors (of lowest indices) and
    assign task i to them (twice).
2)  i <-- i + 1.  If i > n then stop or else go to (1).
```

Here the assumption is that $u_1 \geq \ldots \geq u_n$.

Definition 3.1: Let S be a set of numbers. Define
nin S = min (S - {min S}). This is known as the "next minimum."
N.B. nin T $\leq$ nin S, T $\leq$ S.

Lemma 3.1: Suppose that algorithm A is applied to tasks with utilizations $u_1 \geq \ldots \geq u_n$ on m processors. Let $q_1, \ldots, q_m$ be the resulting processor utilizations. Then $\max_j \{q_j\} - \operatorname{nin}_j \{q_j\} \leq u_k$ where k is the last task assigned by A to the maximally utilized processor.

Proof: Suppose that $q_1 = \max_j \{q_j\}$ and one copy of task k is assigned to processor 1, algorithm A assigns task k to the two least utilized processors. Assume that task k was assigned to processors 1, 1'. Let $p_1, \ldots, p_m$ be the accumulated processor utilizations just immediately before task k was assigned to processors 1, 1' by algorithm A. Then $p_1 + u_k = q_1 = \max_j \{q_j\}$. Also, $p_1 \leq \operatorname{nin}_j \{p_j\}$, since processor 1 was one of the two least utilized processors at the time task k was being assigned. Moreover, $p_j \leq q_j$ $(1 \leq j \leq m)$. Therefore $p_1 \leq \operatorname{nin}_j \{p_j\} \leq \operatorname{nin}_j \{q_j\}$. Hence, rewriting p_1 as $\max_j \{q_j\} - u_k$, results in $\max_j \{q_j\} - u_k \leq \operatorname{nin}_j \{q_j\}$, and the lemma is proved.

Definition 3.2: Let $q = (q_1, \ldots, q_m)$ and $p = (p_1, \ldots, p_m)$ be two sequences such that $q_j, p_j \geq 0, \sum_j q_j = \sum_j p_j$ and $\max_j \{q_j\} \leq 2\min_j \{q_j\}$. Define $K^{(m)} = \operatorname{lub}_{q,p}\{\sum_j (q_j)^2 / \sum_j (p_j)^2\}$. Also define $K = \operatorname{lub}_m \{K^{(m)}\}$.

Lemma 3.2: Suppose that there are n tasks with utilizations $u_1, \ldots, u_n$ which are to be legally (i.e., two copies of each task are assigned to distinct processors) assigned to m processors. Let $q_1, \ldots, q_m$ be the processor utilizations resulting from algorithm A. Let $q_1^*, \ldots, q_m^*$ be the processor utilizations resulting from optimal algorithm O. Then

$$\sum_j (q_j)^2 / \sum_j (q_j^*)^2 \leq K.$$

Proof: Let $q_1 = \max_j \{q_j\}$, and suppose that task k was the last task assigned by algorithm A to processor 1. Also assume that task k was assigned (redundantly) to processor 1'. Let $p_1, \ldots, p_m$ be the processor utilizations immediately before task k was assigned. We have two cases.

Case 1: $p_1 \neq 0$. Then $0 < p_1 \leq p_j > 0$ $(1 \leq j \leq m,\ j \neq 1')$, since processors 1 and 1' were the two least utilized processors prior to the assignment of task k. Thus $p_j \geq u_k$ $(1 \leq j \leq m,\ j \neq 1')$, since each of these processors had a task (of utilization at least u_k) already assigned to it--recall that $p_j > 0$ $(i \leq j \leq m,\ j \neq 1')$. Therefore $\min_j \{q_j\} \geq u_k$ because $q_j \geq p_j$ and $u_k \leq \min_{j \neq 1'} \{q_j\} \leq \min_j \{q_j\}$ (since $u_k \leq p_j \leq q_j$ for $1 \leq j \leq m$, $j \neq 1'$). Clearly $\sum_j q_j = \sum_j q_j^*$. Hence, by Lemma 3.1,

$$\max_j \{q_j\} \leq \min_j \{q_j\} + u_k \leq 2\min_j \{q_j\}.$$

Therefore,

$$\sum_j (q_j)^2 / \sum_j (q_j^*)^2 \leq K^{(m)}.$$

Case 2: $p_1 = 0$. Then $p_j = 0$ $(j \geq 1)$ and $p_j = u_k$ $(j<1)$ because $q_1 = u_k$ is maximal. Find the smallest number t such that there is a processor s which has at least two tasks assigned to it and $\max_{j \geq t} \{q_j\} = q_s$. For example see Figure 3.2.

There are two subcases of case 2.

Subcase 2.1: No such t exists. Then all processors have less than two tasks assigned to them. In this case A has produced an optimal assignment and

$$\sum_j (q_j)^2 / \sum_j (q_j^*)^2 = 1 \leq K^{(m)}.$$

Subcase 2.2: Such a t exists. Suppose task r was the last task assigned to processor s by algorithm A. Then, by Lemma 3.1,

$\max_{j \geq t} \{q_j\} - \min_{j \geq t} \{q_j\} \leq u_r$. Also, $u_r \leq \min_{j \geq t} \{q_j\}$. Hence,

$$\max_{j \geq t} \{q_j\} \leq \min_{j \geq t} \{q_j\} + u_r \leq 2\min_{j \geq t} \{q_j\}.$$

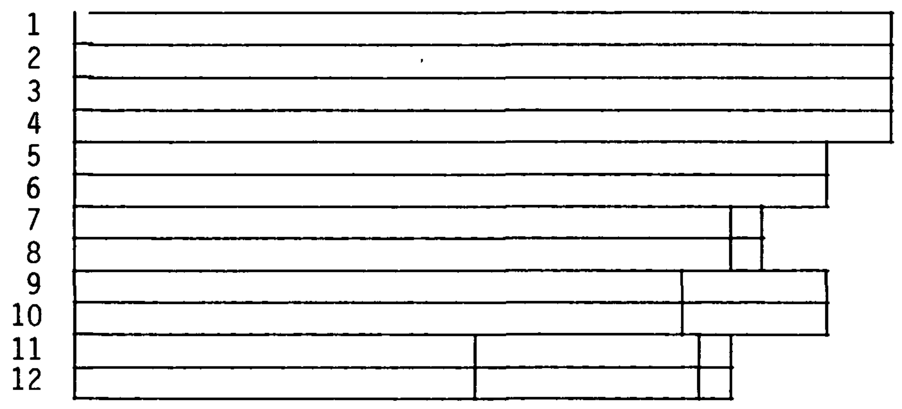
The next step is to perform a cost-preserving transformation on the optimal assignment.

Claim: The optimal assignment resulting in processor utilizations $q_1^*, \ldots, q_m^*$ can be transformed into an (not necessarily legal, i.e., two copies of a task may be assigned to the same processor) assignment with processor utilizations $q_1', \ldots, q_m'$ such that

$$\sum_j q_j^* = \sum_j q_j' , \sum_j (q_j')^2 \leq \sum_j (q_j^*)^2$$

and

$$q_j' = q_j \;(= \; u_{\lceil j/2 \rceil}) \qquad (1 \leq j < t).$$



t = 7 s = 9

Figure 3.2. Example of the construction of t and s.

Proof of Claim: The transformation is as follows:

1) Do until none of the tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$ are co-assigned to the same processor: If two of tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$ are co-assigned to a processor, then place the smaller task on the least utilized processor that has assigned to it none of the tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$. This step is always possible if two of tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$ are co-assigned to a processor. The transformation will never increase the cost of the assignment. (See Figure 3.3.)

2) Assume that now the processors have been renumbered so that task j is assigned to processors 2j-1 and 2j ($1 \leq j \leq \lfloor t/2 \rfloor$). (See Figure 3.4.) Note that none of the tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$ are assigned to processors t, ..., m, and if $1 \leq j_1 \leq j_2 \leq \left\lceil \frac{t-1}{2} \right\rceil$, then tasks j_1 and j_2 are assigned to different processors. Now, if tasks i_1 and i_2 ($1 \leq i_1 \leq \left\lceil \frac{t-1}{2} \right\rceil < i_2 \leq m$) are assigned to processor j_1 ($1 \leq j_1 < t$), then find a minimally utilized processor among t, ..., m (say j_2). Then $q'_{j_2} \leq u_{i_1}$ (by construction of t). Task i_2 can be removed from processor j_1 and assigned to processor j_2 with no cost increase. (See Figure 3.5.)

Step (1) can be applied until each of tasks 1, 2, ..., $\left\lceil \frac{t-1}{2} \right\rceil$ is assigned to a separate processor. Then step (2) can be applied until each (instance) of tasks $\left\lceil \frac{t-1}{2} \right\rceil + 1$, ..., n is assigned to one of the processors t, ..., m. The ultimate assignment is as in Figure 3.6 (after renumbering).

Let $q'_1, \ldots, q'_m$ denote the processor utilizations in the transformed assignment. The transformed assignment has cost no more than the optimal legal assignment; i.e.,

$$\sum_j (q'_j)^2 \leq \sum_j (q^*_j)^2.$$

In both the transformed assignment and the assignment resulting from algorithm A, task i is assigned to processors 2i-1 and 2i ($1 \leq i \leq \lfloor t/2 \rfloor$). Therefore, $q_j = q'_j$ ($1 \leq j < t$), and

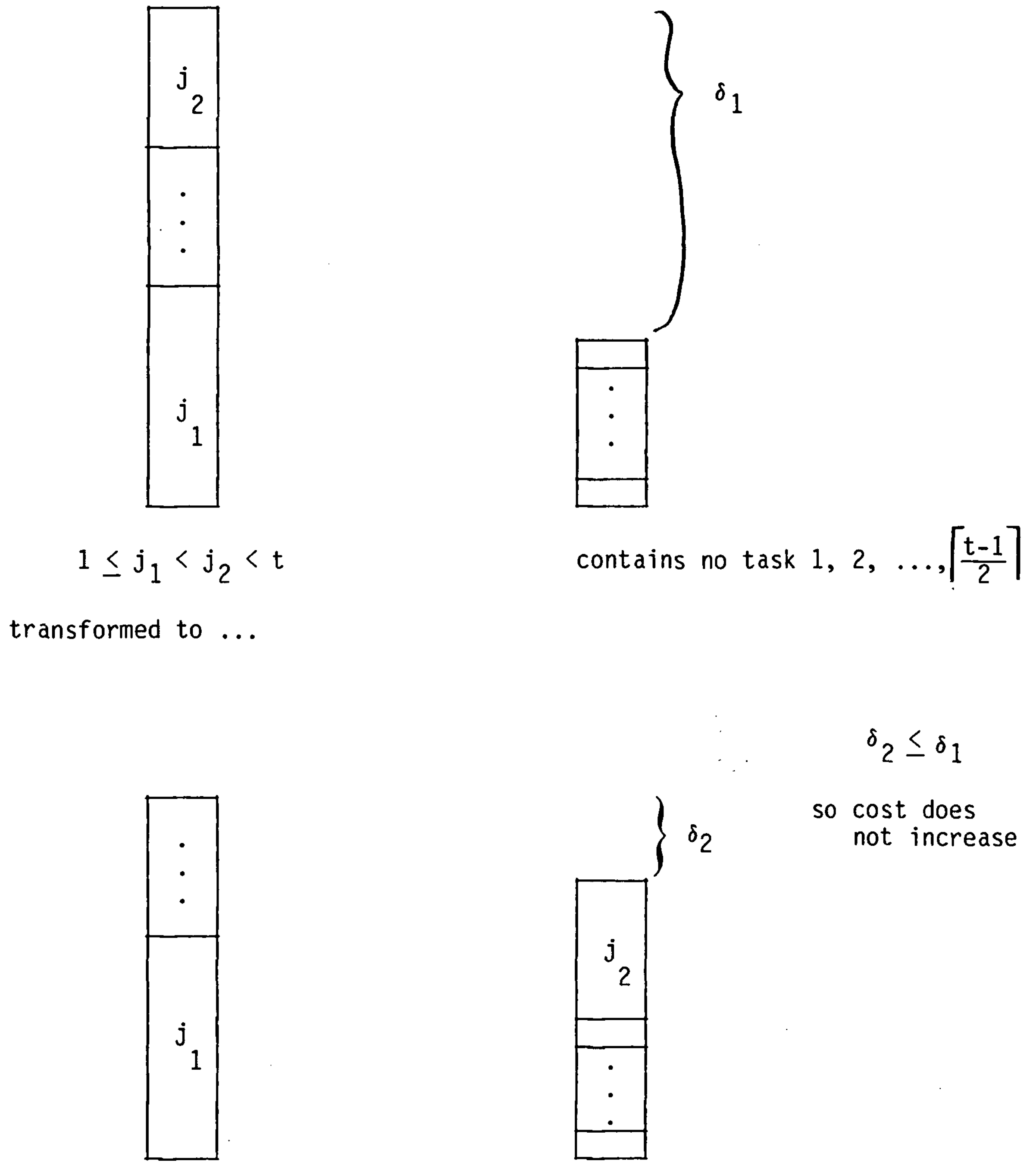


Figure 3.3. Cost-preserving transformation of the optimal assignment by step (1).

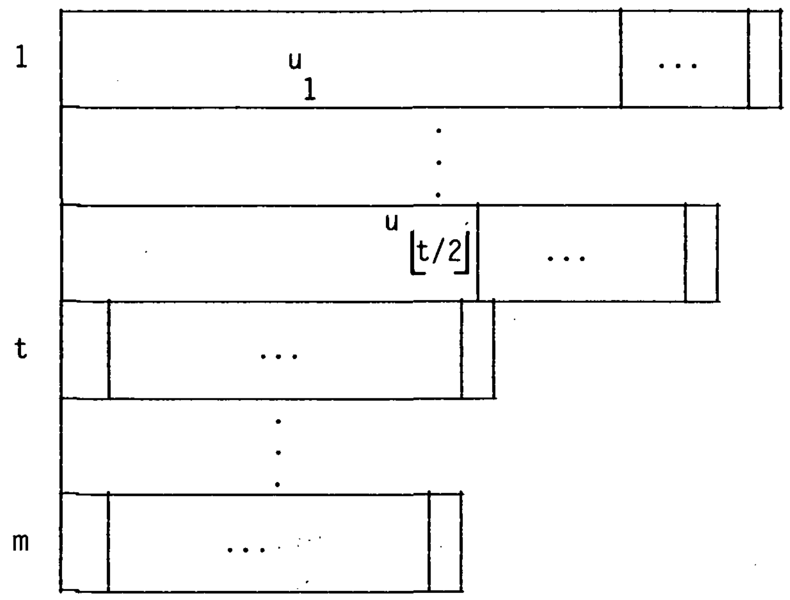


Figure 3.4. Assignment after application of step (1) and renumbering.

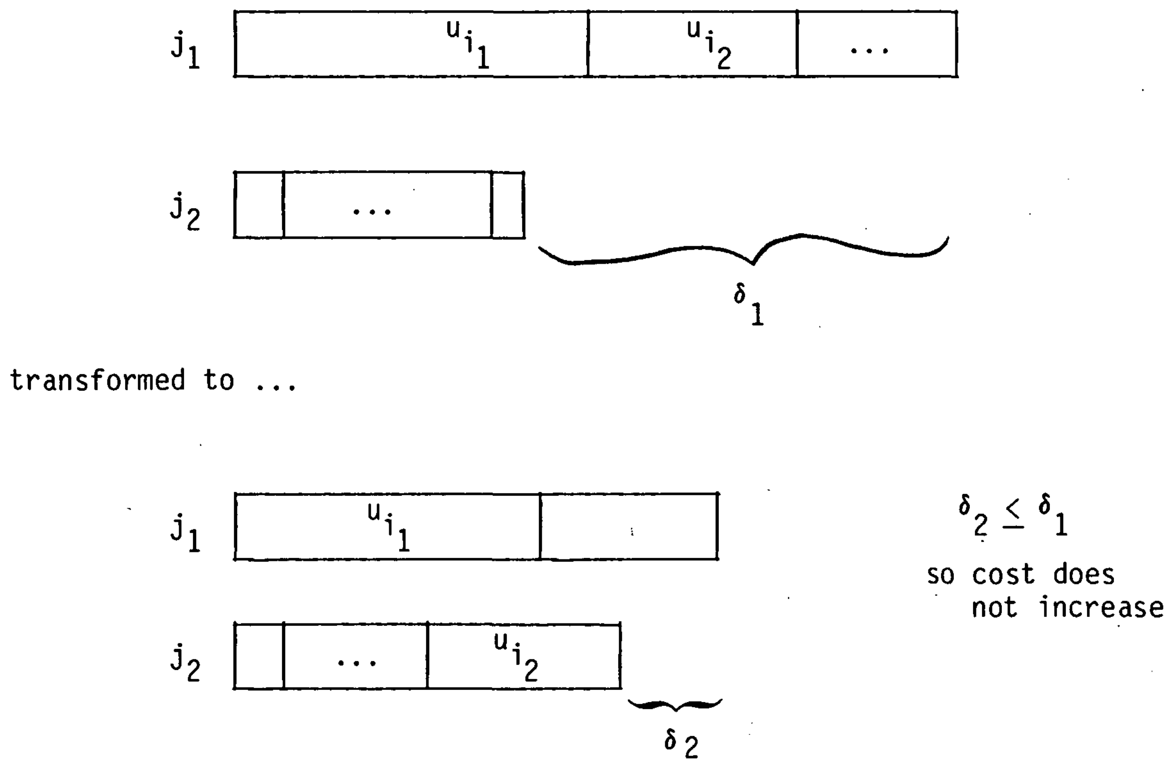


Figure 3.5. Cost-preserving transformation of the optimal assignment by step (2).

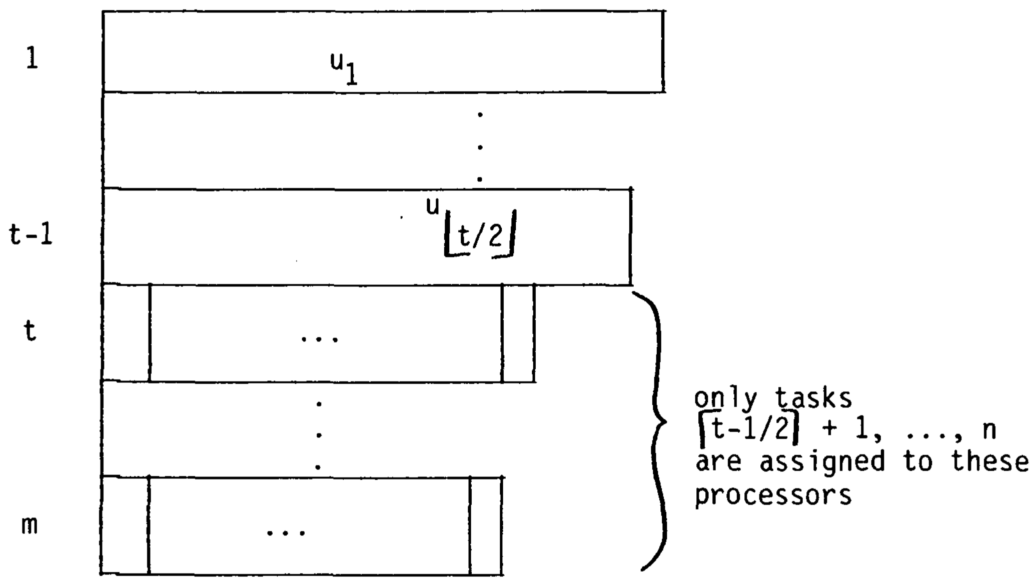

1
u_1
t-1
$u_{\lfloor t/2 \rfloor}$
t
...
m
...
only tasks
$\lceil t-1/2 \rceil + 1, \ldots, n$
are assigned to these
processors

Figure 3.6. Final assignment after transformation by steps (1) and (2). The transformed assignment costs no more than the optimal assignment.

$$\sum_j (q_j)^2 / \sum_j (q_j^*)^2 \leq \sum_j (q_j)^2 / \sum_j (q_j')^2.$$

But, $\sum_j (q_j)^2 = \sum_{j<t} (q_j)^2 + \sum_{j\geq t} (q_j)^2$

and $\sum_j (q_j')^2 = \sum_{j<t} (q_j')^2 + \sum_{j\geq t} (q_j')^2 = \sum_{j<t} (q_j)^2 + \sum_{j\geq t} (q_j')^2.$

Thus, $$\frac{\sum_j (q_j)^2}{\sum_j (q_j^*)^2} \leq \frac{\sum_{j<t} (q_j)^2 + \sum_{j\geq t} (q_j)^2}{\sum_{j<t} (q_j)^2 + \sum_{j\geq t} (q_j')^2} \leq \frac{\sum_{j\geq t} (q_j)^2}{\sum_{j\geq t} (q_j')^2}$$

Also, by the claim,

$$\sum_{j\geq t} q_j' = \sum_{j\geq t} q_j.$$

The task set $\left\lceil \frac{t-1}{2} \right\rceil + 1, \ldots, n$ was assigned to processors $t, \ldots, m$ by algorithm A and the maximally utilized processor s had at least two tasks assigned to it. Therefore, case 1 applies and

$$\max_{j\geq t} \{q_j\} \leq 2 \cdot \text{nin}_{j\geq t} \{q_j\}.$$

Hence, $$\frac{\sum_j (q_j)^2}{\sum_j (q_j^*)^2} \leq K^{(m-t+1)} \leq K.$$

<u>Lemma 3.3</u>: $K^{(m)} \leq \frac{9m}{8(m-1)}$; hence $K < 1.69$ for $m \geq 3$.

Proof: Consider the problem of finding a nonincreasing step function $q(t)$ $(0 \leq t \leq m)$ such that:

1) $q(0) \leq 2q(m - 1)$

2) $\int_0^m q(t)dt = s$ (s a constant)

3) $\int_0^m q^2(t)dt$ is maximized for the given m and s.

If $q_*(t)$ is a nonincreasing step function satisfying conditions (1) - (3), then,

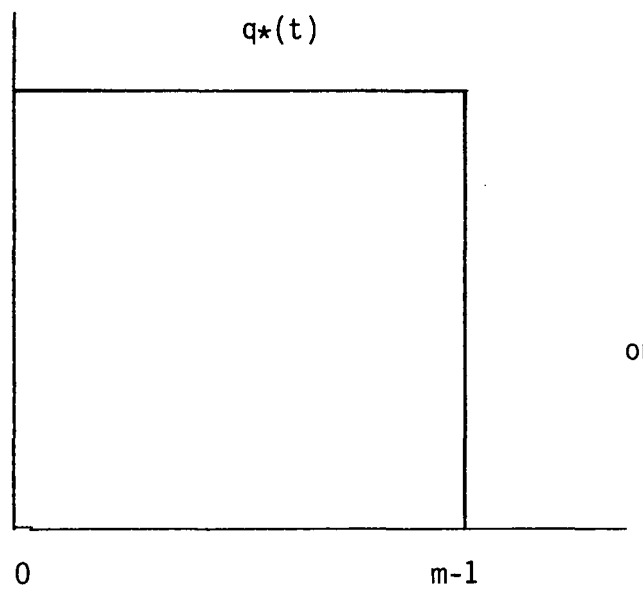
$$K^{(m)} \leq \int_0^m q_*^2(t)dt/(s^2/m)$$

since $\int_0^m q_*^2(t)dt$ represents the maximum value that $\sum_j q_j^2$ can assume where $q_j \geq 0$, $\sum_j q_j = s$ and $\max_j q_j \leq 2\min_j \{q_j\}$ $(1 \leq j \leq m)$; and s^2/m represents the minimum value that $\sum_j p_j^2$ can assume where $\sum_j p_j = s$ and $p_j \geq 0$ $(1 \leq j \leq m)$.

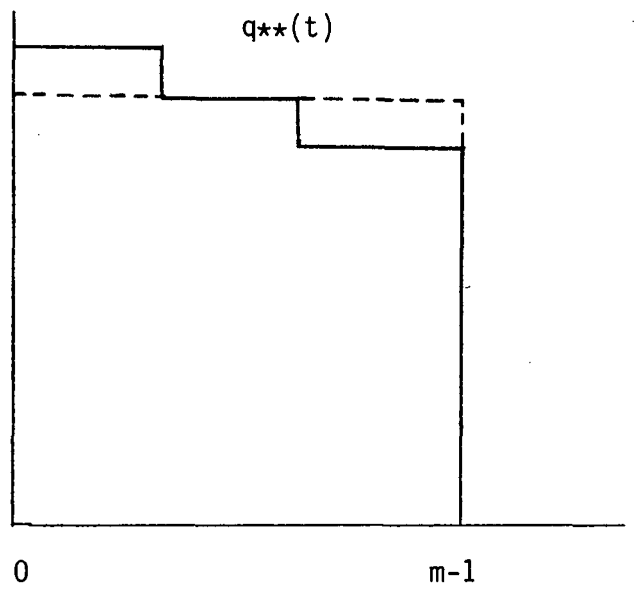
Note that $q_*(t)$ can have only two nonzero "steps." If it had more or fewer, then $\int_0^m q_*^2(t)dt$ would not be maximal. This is illustrated in Figure 3.7.

Therefore, $q_*(t)$ must have the following form.

$$q_*(t) = \begin{cases} 2x & 0 \leq t < k \\ x & k \leq t \leq m-1 \\ 0 & m-1 < t \leq m \end{cases}$$

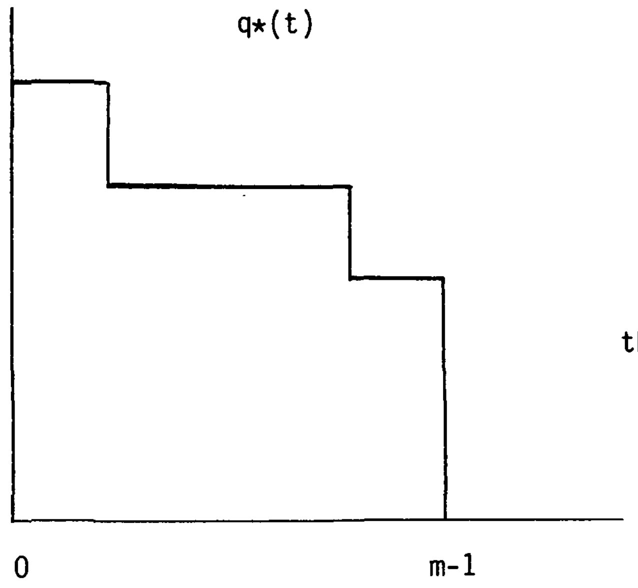


q_* can be transformed to better q_{**}



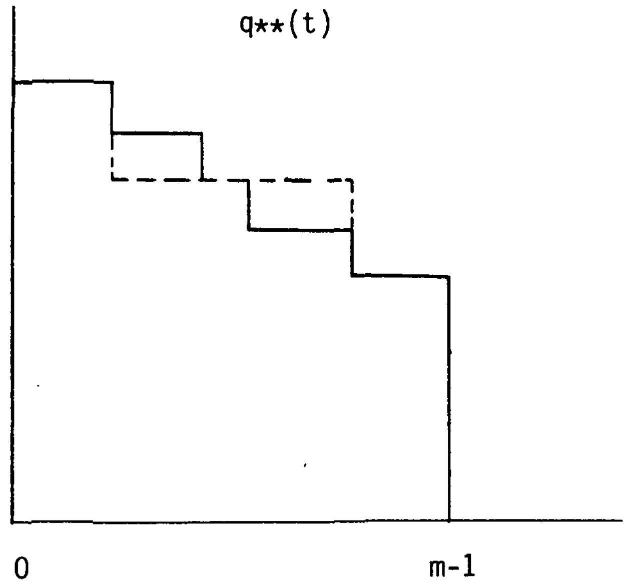
$$\int_0^m q_{**}^2(t)dt > \int_0^m q_*^2(t)dt$$

Figure 3.7. $q_*(t)$ cannot have one step.



three-or-more step

$q_{\star}$ can be transformed to better $q_{\star\star}$



$$\int_{0}^{m} q_{\star\star}^{2}(t)dt > \int_{0}^{m} q_{\star}^{2}(t)dt$$

Figure 3.7 (continued). $q_{\star}(t)$ cannot have three or more steps.

Thus, $s = 2xk + x(m - 1 - k)$ and $k = s/x - m + 1$. So

$$\int_0^{m_2} q_*(t)dt = 4x^2(s/x - m + 1) + x^2(2m - s/x - 2)$$

$$= 3sx + 2x^2 - 2mx^2.$$

This attains a maximum of

$$\frac{9s^2}{8m - 8} \text{ when } x = \frac{3s}{4m - 4}.$$

Hence, $K^{(m)} \leq \frac{9}{8} \bullet \frac{m}{m - 1}$.

For the PUBP the performance of approximation algorithms is compared to optimal algorithms. Simply comparing the statistical variance is not sufficient since one should normally expect the variance allowances to differ for differing task sets. Therefore a measure akin to the coefficient of variation will be used. Given an approximate assignment (x_{ij}) and an optimal assignment (x'_{ij}) the relative costs are compared as follows:

$$\frac{\frac{1}{m}\sum_j\left(\sum_i x_{ij}\, u_{ij}\right)^2 \Big/ \left(\sum_j\sum_i x_{ij}\, u_{ij}/m\right)^2}{\frac{1}{m}\sum_j\left(\sum_i x'_{ij}\, u_{ij}\right)^2 \Big/ \left(\sum_j\sum_i x'_{ij}\, u_{ij}/m\right)^2}.$$

Theorem 3.1: Suppose that there are n tasks with utilizations $u_1, \ldots, u_n$ which are to be legally (i.e., two copies of each task are assigned to distinct processors) assigned to m processors. Let $q_1, \ldots, q_m$ be the processor utilizations resulting from algorithm A. Let $q_1^*, \ldots, q_m^*$ be the processor utilizations resulting from optimal algorithm 0. Let $V_A = \frac{1}{m}\sum_j q_j^2 \Big/ \left(\frac{2}{m}\sum_i u_i\right)^2$ denote the relative variation in processor utilization under the approximate assignment A. Let $V_0 = \frac{1}{m}\sum_j (q_j^*)^2 \Big/ \left(\frac{2}{m}\sum_i u_i\right)^2$ denote the relative variation in processor utilization under the optimal algorithm 0. Then

$$V_A \Big/ V_0 \le \begin{cases} 1 & (m = 2) \\ \frac{9m}{8(m-1)} < 1.69 & (m \ge 3) \end{cases} .$$

Hence, the approximation algorithm is always less than 69% off optimum.

Proof: Note that $V_A \Big/ V_0 = \sum_j q_j^2 \Big/ \sum_j (q_j^*)^2$.

But for m = 2, algorithm A produces optimal assignments. Therefore, by Lemma 3.3, $V_A \Big/ V_0 \le \frac{9m}{8(m-1)} < 1.69$ for $m \ge 3$, and $V_A/V_0 = 1$ for m = 2.

3.1.3 Task allocation for reliability maximization*. - A function $f(\underline{x})$ of a vector argument $\underline{x} = (x_1, x_2, \ldots, x_n)$ is said to be symmetric if for any permutation $\underline{y} = (y_1, y_2, \ldots, y_n)$ of $\underline{x}$, $f(\underline{x}) = f(\underline{y})$. A function $f(\underline{x})$ is said to be concave if

$$\lambda f(\underline{x}^{(1)}) + (1-\lambda) f(\underline{x}^{(2)}) \le f(\lambda \underline{x}^{(1)} + (1-\lambda)\underline{x}^{(2)}).$$

Definition of a convex function is obtained by reversing the inequality above. A set C of points is said to be convex if for

$$\underline{x}^{(1)}, \underline{x}_2^{(2)} \epsilon C, \lambda \underline{x}^{(1)} + (1-\lambda)\underline{x}_2^{(2)} \epsilon C \quad \text{for } \lambda \ge 0.$$

*Section 3.1.3 was contributed by Kishor Trivedi, Duke University.

Theorem 3.2: A symmetric concave function $f(\underline{x})$ over a symmetric convex space is maximized by equalizing all of its arguments.

Proof: Assume that there is a two-dimensional feasible space for convenience, although the argument is easily generalized. The proof proceeds by contradiction.

Assume that the optimal point is (x,y) where $x \neq y$. Since the feasible space is symmetric, then (y,x) is also in it. Furthermore, since the feasible space is convex, any point on the line joining the two points is also feasible. In particular, $(\frac{x+y}{2}, \frac{x+y}{2})$ is feasible. By symmetry of $f(\underline{x})$, $f(x,y) = f(y,x)$ is the maximal value, but by concavity of f,

$$f(\frac{x+y}{2}, \frac{x+y}{2}) \geq \frac{1}{2}f(x,y) + \frac{1}{2}f(x,y) = f(x,y)$$

which implies that $(\frac{x+y}{2}, \frac{x+y}{2})$ is also an optimum.

Thus, if one assumes that the system reliability is a concave function of detection latencies, for instance, then clearly it being symmetric, the reliability is maximized by equalizing detection latencies. Similarly, reliability is maximized by equalizing reconfiguration times and so on. Given enough time, concavity of reliability may also be established.

3.2 Discussion of the Role of Software Design*

3.2.1 Introduction to classical engineering design methodology. - This section discusses the elements of classical engineering design methodology with particular reference to software engineering. It assumes a basic familiarity with the problems of software engineering.

*Section 3.2 was contributed by Thomas Alspaugh, Jr., IBM, Research Triangle Park.

The term "methodology" is often considered a buzzword. According to Webster, a methodology is "a body of methods, rules, and postulates employed by a discipline". Sage (See ref. 29) narrows it to "a set of tools, a set of proposed activities for problem solution, and a set of relations among the tools and activities". These are very broad, high-level definitions; indeed, methodologies themselves are difficult to express except in broad, high-level terms. They are so abstract that a true understanding of them requires a consideration of the concrete application which inspired them.

Engineering solutions can be applied to a certain restricted class of problems that can be adequately expressed in explicit, concrete, and quantitative terms. Such solutions are characterized by predictable behavior which can be quantified and verified. The creation of such solutions is constrained by the limited capacity to deal with complex ideas or consider all the ramifications of complex interactions. The methodology deals with these issues and problems.

Nearly everyone has had the experience of apprehending and solving a simple everyday engineering problem almost at one stroke, with no conscious effort. The problem was apprehended intuitively, synthesis of the solution occurred unconsciously, and understanding of the situation was so intimate that it was a foregone conclusion that the realization of the solution would meet the need completely. But most problems are not so tractable. The methodology helps one to solve problems by conscious effort while trying to attain the unity and integrity that characterize the correct intuitive solution.

3.2.2 The classical engineering design methodology. - The methodology consists of a number of components: abstraction, the "black box", specifications and testing, the design process, iterative convergence, recursive decomposition, documentation, and pilot operation. Each component is considered below.

3.2.2.1 Abstraction: Abstraction is a powerful concept that permeates the engineering design methodology. The process of abstraction consists of conceptually separating the important characteristics of a situation or object from a mass of unimportant information. The characteristics that were abstracted form a model or abstraction that can stand for the situation or object itself to a certain extent. This abstraction can be studied and manipulated instead of the complex thing from which it was abstracted, and if the abstracted characteristics are well chosen the results are applicable to the thing itself. This is a powerful technique for dealing with complex things and ideas.

Three examples of the use of abstraction in the creation of engineering solutions follow. In considering the problem, certain aspects of it will be important and others less so. Also, it may be useful to consider some characteristics of the problem in terms of mathematics or some other mental model rather than in the terms of the problem itself, thus ignoring some aspects of those characteristics. The solution itself will have many properties irrelevant to the problem it solves, or it may be only an approximate solution, though close enough for practical purposes.

3.2.2.2 The black box: The concept of the black box is a particularly important example of abstraction. A black box is a system whose inner workings are completely invisible. Its externally visible behavior can be described completely and concisely. Its internal structure may be, and probably is, much more complex, but this complexity need not be considered in using the black box. An engineering solution that can be conceived of as a black box is more useful than one that cannot. Such a system is allowed more flexibility in its internal workings, since only a small subset of its characteristics are constrained. It is easier to think about and use, since its behavior can be described more abstractly and more concisely. It is easier to reuse later in another problem, since it does not require a large investment of time to understand how it behaves.

3.2.2.3 Specifications and testing: The technique of specifications and testing arises from the nature of engineering, which considers things in terms of quantifiable physical characteristics. Thus, an existing or desired engineering solution can be adequately described in such terms. An abstraction of a solution or a set of solutions can be so expressed; if the abstraction contains precisely the properties that a satisfactory solution must have, then its description constitutes the specifications for the solution, the requirements which a solution must meet in order to be satisfactory. The properties of a candidate solution can be tested and compared with the specifications to decide if the candidate is a satisfactory solution.

3.2.2.4 The design process: The design process is a sequence of steps whose goal is to develop a solution for a complex problem. The process is arranged to encourage the same integrity that occurs in the case of intuitive solutions to simple problems. The steps of the process are depicted in Figure 3.8. They are:

1. Problem definition. The problem is explored and its important characteristics are abstracted. No reference is made to possible solutions; the problem is considered in its own terms. Objectives and criteria for solutions are developed and the worth of having a solution (or the cost of not having a solution) is estimated.

2. Requirements. The statement of the problem is recast in terms of engineering. The requirements that a candidate solution will have to meet in order to be acceptable are developed. Still no reference is made to specific solutions.

3. Design synthesis. A design for a system that will meet the requirements is synthesized.

4. Realization. The design is realized.

5. Design test. The realization of the design is tested to ensure that it meets the requirements determined in the second step.

6. Requirements test. The realization of the design, which has now been established as an embodiment of the requirements, is tested in the context of the problem to ensure that the requirements meet the definition of the problem and that the definition of the problem is accurate.

The integrity of the process is verified by the testing steps. Divergence can occur at each point at which a new abstraction is developed: problem definition, establishment of requirements, design synthesis; and at the point at which an abstraction is realized: design realization. The process may be viewed as a chain of abstractions leading from the problem to its concrete solution. If each abstraction is accurate and well chosen

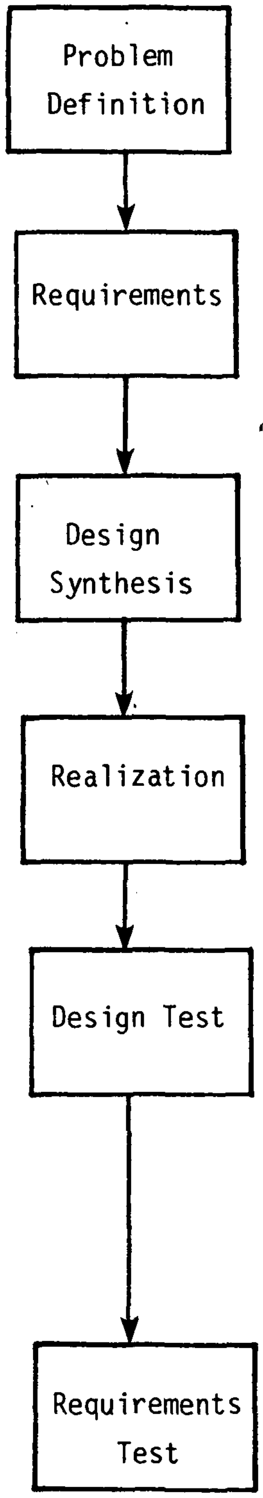


Figure 3.8. Steps in the Design Process.

and if the realization correctly reflects the final abstraction, then the solution will necessarily meet the needs of the problem. The testing ensures that each link in the chain is firmly joined to what comes before.

3.2.2.5 Iterative convergence: Iterative convergence refers to the trial-and-error process by which the abstractions in the design process are improved. The limitations of our mental capacity make it unlikely that abstractions will be adequate from the start. The testing steps may reveal faulty abstractions and provide a means for pinpointing their shortcomings. The design sequence is then followed again, starting from the improved abstraction (Figure 3.9).

3.2.2.6 Recursive decomposition: Nearly all problems are too large to solve as a whole. The required solution would be too complex to be adequately understood. In such cases the solution can be analyzed as the combination of several black boxes, each of which is then considered as a subproblem. The subproblems are treated precisely as any other problem needing an engineering solution; each subproblem is analyzed, requirements are generated, and a subsolution is designed, realized, and tested. If these subproblems are also too large, they can be similarly decomposed in a recursive manner until manageable problems are obtained. The solution to the original problem is obtained by recursively assembling all the subsolutions.

The correctness of the abstraction involved in the decomposition is tested when the subsolutions have been implemented by combining them and testing that the combination meets the requirements established for the solution.

3.2.2.7 Documentation: Documentation is an important part of the creation of an engineering solution in that it provides information needed by the users and maintainers of the solution. Less obvious is its value to the creators of the solution. The structure of the documentation directs the thought of the writer, and if the structure of the documentation is well chosen, the writer will be assisted in relating the important issues of the solution. The act of writing itself forces the thorough consideration of what is being written, and questions that can be glossed over in

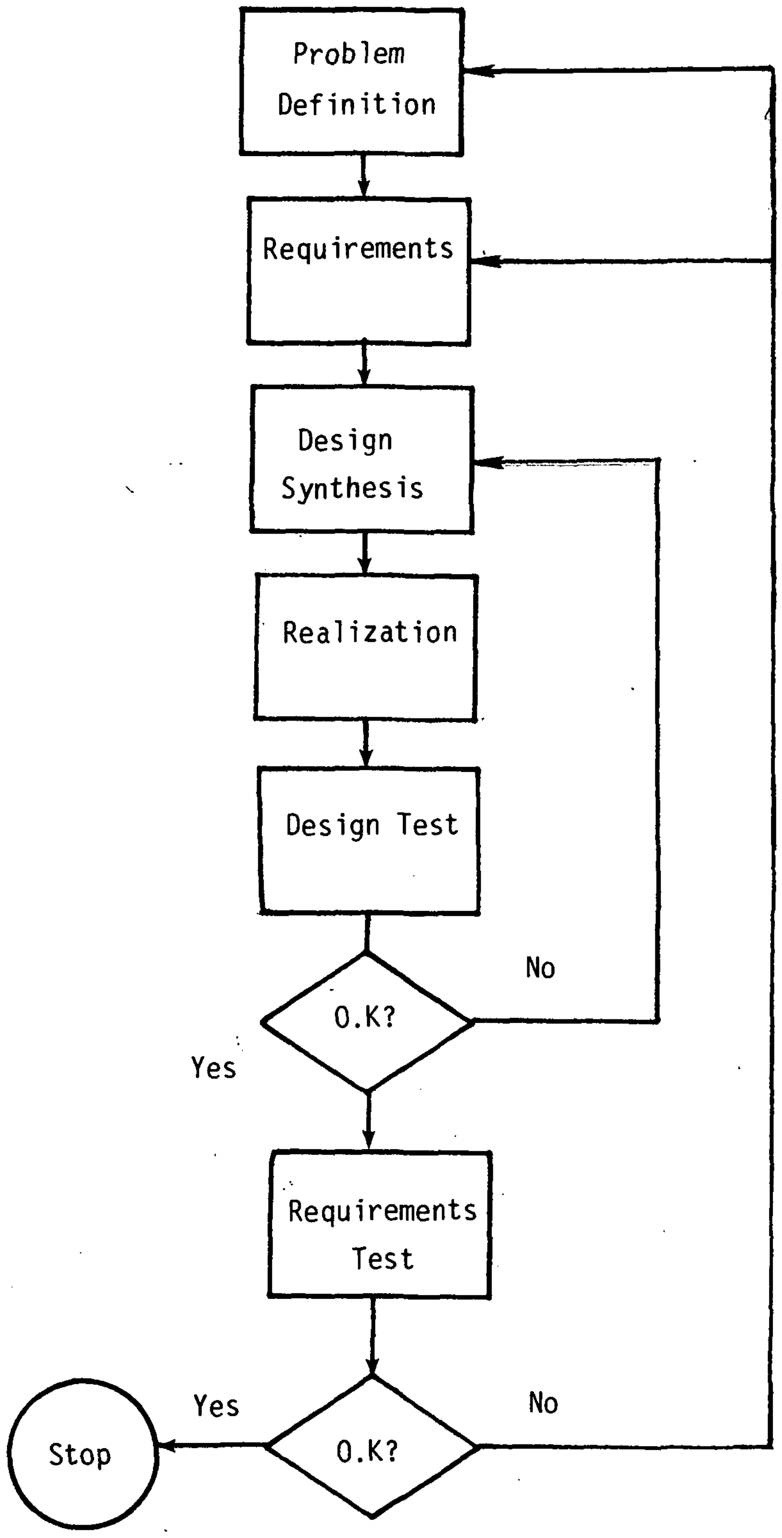


Figure 3.9. Iterations for Improving Abstractions.

one's mind become painfully obvious when they are put on paper. Brooks (See ref. 4) also points out that the management of the creation of an engineering solution revolves around a relatively small body of information concerning the system, and the writing down and dissemination of this information is a crucial part of the managing task.

3.2.2.8 Pilot operation: The term "pilot operation" refers to a small-scale test solution used to gain knowledge and experience that will be useful in the creation of a large and expensive solution. The pilot operation risks a relatively small investment to reduce the risk that the large investment for the actual solution will be wasted. Pilot operations are useful for solutions embodying ideas that have never been tested or that have been tested only on a much smaller scale; large changes of scale change the relative importance of the aspects of the problem and solution which must be taken into account, thus possibly invalidating the abstractions that were appropriate on a small scale.

3.2.3 Software engineering. - Certain areas of the methodology pose special problems or are of special interest in software engineering. They are described below, organized by the component of the methodology with which they are concerned.

3.2.3.1 Abstraction: In Brooks' phrase, programmers work with a very tractable medium (See ref. 4). The flexibility inherent in programming means that any program is the embodiment of a large number of implicit or explicit decisions which together define the program. It is difficult to abstract from such a large number of decisions, particularly since they may interact in complex ways. Also, software deals with dynamic processes that evolve over a period of time; such processes are more difficult to think about than the more static relations that characterize other engineering disciplines (See ref. 11).

3.2.3.2 Black box: For these same reasons, black boxes are more of a problem in software engineering. Problems with software engineering black boxes are usually thought of as problems in the interface, which is the manifestation of the abstraction that describes the black box. Software black boxes also are less easily portable because the inputs they require may be difficult to abstract and understand.

3.2.3.3 Specification and test: Software specifications commonly specify both too little, and in one sense, too much. The large number of important decisions makes it difficult to cover all the important decisions when making specifications; this leads to the ad hoc making of design decisions by the implementers, who usually have no overview of the solution and cannot be relied upon to choose correctly. In another sense, specifications often specify too much, in that they unduly restrict the implementation (See ref. 23).

3.2.3.4 The design process: Several parts of the design process are of particular concern.

When considering a problem to be solved by software, it is very tempting to define the problem itself by specifying to some level of detail a possible solution. Most software engineers find it easier to think in software terms than in terms of a specific problem.

Two aspects of the process of synthesizing the design of a solution are of special concern. First, it is very desirable but not easy for designs to reflect the natural structure of a solution. It is desirable because such designs are much easier to think about, but it is difficult because problems almost always have some aspects that are difficult to reconcile with the main characteristics. Second, solutions usually need to be adapted to slightly different problems as time goes by, and it is desirable to design a solution so it may be adapted easily.

Testing software is of limited value since tests can only show the presence of errors, not their absence. The complexity of software solutions requires extensive tests to provide confidence that the software is reasonably free of errors. Formal proof is a tempting alternative, but it has drawbacks also. The process of proving correctness is arduous, and also it is all too easy to prove one thing while thinking one is proving something else.

3.2.3.5 Iterative convergence: The iterative process of testing, finding, and fixing problems in a software solution tends over time to make the structure of the solution less orderly, particularly if the problems show up some length of time after the solution was tested and declared accurate. The temptation to attempt to repair the solution without taking the time to understand it completely, together with the natural human tendency to treat symptoms before the underlying problem is evident, combine to make the solution not fit its abstractions and behave in more complex, less predictable ways as it grows older.

3.2.3.6 Documentation: The complexity of software solutions makes them more difficult to document. Because software is so flexible and tractable, any solution involves a large number of decisions. The amount of information that must be organized and recorded in the documentation makes it likely that the organization of that information will not be very good. Ideally, the documentation should be useful both as an introduction or overview for a neophyte and as a reference for a person who has at least the knowledge provided by the overview but not necessarily a thorough understanding of the whole software system. In fact, this rarely occurs.

Parnas (See ref. 24) lists six important problems plaguing most software documentation.

1. It is obtuse in the assumptions made about the purposes and knowledge of the reader.
2. It is hard to use as a reference.
3. It is expensive to maintain.
4. It is repetitive and boring, and thus is not read attentively.
5. It uses inconsistent terminology.
6. It is written for people who know as much as the writers of the documentation.

The writers of most documentation try to make each piece of information understandable when looked up for reference by surrounding it with the background they believe it requires. This means that any particular fact is likely to be recorded in many places throughout the documentation, and if the decision leading to that fact is changed a great deal of effort is required to revise the documentation. It also means that a person reading the documentation will encounter many facts over and over again, probably expressed in exactly the same sentences and paragraphs.

3.2.4 Software engineering tools and techniques. - Certain tools and techniques have been developed to aid in the creation of software engineering solutions.

3.2.4.1 A set of documents: Brooks (See ref. 4) presents a set of documents intended to aid in the management of a software project. They consist of: (1) the objectives--defining the problem, the goals, the constraints and the priorities; (2) the specifications--consisting of the user's manual for the system to be created, which he views as setting forth the requirements for the visible behavior of the system, and the internal documentation; (3) the schedule; (4) the budget; and (5) the space allocation and organizational chart for the people working on the system.

Parnas (See ref. 24) lists a set of documents useful in the creation of a software system. They are: (1) the requirements for the solution; (2) secret test plans, to be concealed from the implementers of the solution; (3) the software decomposition document; (4) the individual module interface documents; (5) the individual module design documents; (6) the uses/subsets document, setting forth the interdependencies between modules and the useful subsets of the whole system; and (7) the process structure document, dealing with scheduling.

3.2.4.2 Formal languages: English prose is not well adapted for a precise and unambiguous description of software systems. Many formal languages have been developed for various aspects of software engineering, though none has been very widely used except in the specification of programming languages. Two examples of formal languages which bear on the software engineering design methodology are TRACES, which is used in writing module specifications (See ref. 2), and SPECIAL, which is used in writing more general specifications (See ref. 25).

3.2.4.3 Modularity and information hiding: The classical motivation behind the use of black boxes is portability. Two more motivations for modularization are the splitting of the task into work assignments for a single person or a small group so that a minimum of coordination and communication between subtasks is needed, and the reduction of the interdependence of the decisions that together define the solution, so that at least some decisions may be changed without reworking much of the solution

(See ref. 23). To achieve these goals, the system may be modularized using Parnas' principle of information hiding, so that every decision that is likely to change is hidden in an abstract module whose external behavior is not dependent on the decision, and that in fact every module in the system depends not at all or as little as possible on the way any particular module is implemented.

3.2.4.4 Reference vs. statement in documentation: In computer languages an abstract name can be used to refer to a constant whose actual value is defined at precisely one place in a program. Documentation written analogously by using abstract names to refer to specific decisions would be very simple to change; if the documentation was organized strictly and simply, it would also be easy to use as a reference. Heninger et al. (See ref. 17) have done precisely that in the complete rewriting of the very large requirements document for the A-7 Operational Flight Program, and the result has been quite successful.

3.2.4.5 Simulation: Software engineering allows another kind of decomposition. A solution that would normally be a complex program running on a simple machine can be broken down into a simpler program running on a more complex abstract machine, and a program, running on the simple machine, which simulates the abstract machine. A well-chosen decomposition of this type can make a significant reduction in the effort required to implement a solution.

4.0 CONCLUSIONS AND FURTHER WORK

This report has presented a methodology for the initial identification and design of integrated aircraft electronic systems that are intended for use in commercial aircraft.

The report presented a taxonomy of multiprocessor systems as a framework for understanding, discussing and characterizing systems. This taxonomy included the identification of major system design parameters. A methodology for estimating system behavior was also proposed. System behavior is partially described in terms of the reliability, performance and cost of a system. Measures for reliability, performance and cost were defined and applied to the proposed taxonomy, and a strawman system was identified.

The report also included a discussion of algorithms for and control of fault-tolerant integrated aircraft electronic systems. Management of system resources, especially scheduling and task allocation were shown to be important to the reliability and performance of a fault-tolerant integrated aircraft electronic system. Formal analysis of scheduling and allocation algorithms demonstrate the usefulness of approximation or heuristic algorithms. The role of software design methodologies was discussed in the context of integrated aircraft electronic system design.

4.1 Additional Tools Required

Various tools will aid the development of a methodology for designing fault-tolerant integrated aircraft electronic systems. Two classes of tools are needed.

1) Tools to aid in the classification and description of multiprocessor systems. This includes hardware description languages and design capture (display) systems.

2) Tools to aid in system simulation and in the estimate of identified system measures.

4.2 Techniques and Theoretical Work Needed

A fully integrated aircraft electronic system enhanced by fault-tolerant hardware and software will represent one of the most complex systems ever engineered. Large gaps still exist in our knowledge of fault-tolerant systems and multiprocessor systems. These gaps cut across the areas of hardware design and especially software design. Further theoretical work is suggested in the following areas:

1) Develop the theory of fault tolerance to the point where it can be extended to the design of sensor-actuator systems.

2) The study of multiprocessor systems is still in its infancy. Bring this field of study to the point of maturity. It is recommended that particular emphasis be placed on how to make these systems fault-tolerant and on how to efficiently manage resources and contain complexity.

3) There is a glaring need to be able to identify precisely the functional scope of a projected system. Even more critical is the need to pair the functional elements of a projected system with reliability and performance. Once this has been done, there is the need to translate the function specifications and requirements into an actual hardware-software system.

4) Each of the findings pertaining to (1) - (3) should be based on and substantiated by experimental data and actual experience.

4.3 Peer Review of Specific Systems

Perhaps the most effective way to design and evaluate systems is by exposing the system design to stone throwing at various points in the design of the system. Working groups, conferences and frequent publication of one's efforts provide an ideal forum for such review.

REFERENCES

1. Anderson, G.A., and Jensen, E.D.: Computer Interconnection Structures: Taxonomy, Characteristics and Examples, ACM Computing Surveys, v. 7, n. 4, December 1975.

2. Bartussek, W., and Parnas, D.L.: Using Traces to Write Abstract Specifications for Software Modules, Technical Report TR77-012, University of North Carolina at Chapel Hill, December 1977.

3. Bell, C. G., and Newell, A. Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971.

4. Brooks, F.P., Jr.: The Mythical Man-Month, Reading, MA: Addison-Wesley Publishing Co., 1975.

5. Buckles, B., and Hardin, D.: Partitioning and Allocation of Logical Resources in a Distributed Computing Environment, Tutorial: Distributed System Design, IEEE, New York, 1979.

6. Byrne, J., and Proll, L.: Algorithm 341, Solution of Linear Programs in 0-1 Variables by Implicit Enumeration, CACM, v. 13, n. 4, April 1970.

7. Coffman, E.G., Jr.: Computer and Job Shop Scheduling Theory, John Wiley & Sons, New York, 1976.

8. Davis, A.L., Denny, W.M., and Sutherland, I.: A Characterization of Parallel Systems, Technical Report UUCS-80-108, Department of Computer Science, University of Utah, August 1980.

9. Dhall, S.K.: Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems, Ph.D. Dissertation, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1977.

10. Dhall, Sudarshan K., and Liu, C.L.: On a Real-Time Scheduling Problem, Operations Research, v. 26, n. 1, January-February 1978.

11. Dijkstra, E.: Go to Statement Considered Harmful, Communications of the ACM, v. 11, n. 3, pp. 147-148, March 1968.

12. Flynn, M.J.: Some Computer Organizations and Their Effectiveness, IEEE TC, v. C-21, n. 9, September 1972.

13. Garey, M.R., and Johnson, D.S.: Computers and Intractability, W.H. Freeman, San Fransisco, 1979.

14. Graham, R.L.: Bounds on Multiprocessing Timing Anomalies, SIAM J. Appl. Math, v. 17, n. 2, March 1969.

REFERENCES
(Continued)

15. Gylys, V.B.,and Edwards, J.A.: Optimal Partitioning of Workload for Distributed Systems, Tutorial: Distributed System Design, IEEE, New York, 1979.

16. Han, Y.W.: Performance Evaluation of a Digital System, Using a Petri Net-like Approach, Proceedings of the National Electronics Conference, v. 32, October 1978.

17. Heninger, K.L., Kallander, J.W., Parnas, D.L., and Shore, J.E.: Software Requirements for the A-7E Aircraft, Washington, D.C.: NRL Report 3876, Naval Research Laboratory, Nov. 27, 1978.

18. Kini, Vittal: Automatic Generation of Reliability Functions for Processor-Memory-Switch Structures, Ph.D. Thesis, Carnegie-Mellon University, February 1981.

19. Kosy, Donald W.: The ECSS II Language for Simulating Computer Systems, Rand, Santa Monica, Ca., December 1975.

20. Land, A., and Powell, S.: Computer Codes for Problems of Integer Programming, Annals of Discrete Mathematics, v. 5, 1979.

21. Lawler, E.L., and Martel, C.U.: Scheduling Periodically Occurring Tasks on Multiple Processors, IPL, Feb. 1981.

22. Liu, C.L., and Layland, J.W.: Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment, JACM, v. 20, n. 1, January 1973.

23. Parnas, D.L.: On the Criteria to Be Used in Decomposing Systems into Modules, Communications of the ACM, v. 15, n. 12, pp. 1053-1058, Dec. 1972.

24. Parnas, D.L., Hester, S. D., and Utter, D. F., "Using Documentation as a Software Design Medium," BSTJ, v. 60, n. 8, Oct. 1981, pp. 1941-1977.

25. Robinson, L.: The HDM Handbook: The Foundations of HDM, v. 1, Technical Report for NOSC Contract N00123-76-C-0195, June 1979.

26. Ramamoorthy, C.V.: The Design Methodology of Distributed Computer Systems, NTIS Report AFOSR-TR-80-0542, June 1979.

27. Rao, G.S., Stone, H.S., and Hu, T.C.: Assignment of Tasks in a Distributed System with Limited Memory, IEEE TC, v. C-28, n. 4, April 1979.

REFERENCES
(Continued)

28. Ross, G., and Soland, R.: Modeling Facility Location Problems as Generalized Assignment Problems, Management Science, v. 24, n. 3, 1977.

29. Sage, A.P.: A Case for a Standard for Systems Engineering Methodology, IEEE Transactions on Systems, Man, and Cybernetics, v. SMC-7, n. 7, pp. 499-504, July 1977.

30. Siewiorek, D., Bell, C.G., and Newell, A.: Computer Structures: Principles and Examples, McGraw-Hill, New York, 1982.

31. Taha, H.: Integer Programming: Theory, Applications, and Computations. Academic Press, New York, 1975.

32. Wensley, J.H., et al.: SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control, IEEE Proceedings, v. 66, n. 10, October 1978.

33. Wensley, J.H., et al.: Design Study of Software-Implemented Fault-Tolerance (SIFT) Computer, Interim Technical Report for NASA Contract NAS1-13792, June 1978.

Appendix A

The following is a listing
of the original heuristic algorithm
discussed in Section 3.1.2.3.

```
C       THIS IS A PROGRAM FOR HEURISTIC APPROACH OF ALLCATING NT TASKS
C       WITH IR REPLICATIONS TO MP PROCESSORS SO THAT DIFFERENT
C       REPLICATIONS ARE ALLOCATED TO DISTINCT PROCESSORS AND
C       PROCESSOR UTILIZATION IS BALANCED
C       PU ARRAY KEEPS TRACK OF UTILIZATION OF PROCESSORS
C       IORDER KEEPS THE ORDER OF PROCESSOR UTILIZATION ARRAY
C       WHEN SORTED IN ASCENDING ORDER
        REAL*4 U(40),PU(12)/12*0.0/,B(12)/12*0.0/
        INTEGER IORDER(12)
        READ(1,801) IR,NT,MP
801     FORMAT(3I5)
C       IR=NUMBER OF REPLICATIONS PER TASK
C       NT=NUMBER OF TASKS
C       MP=NUMBER OF PROCESSORS
        WRITE(3,901) NT,IR,MP
        MAX=NT
        N=0
10      READ(1,802) X
802     FORMAT(F7.3)
                IF (X.EQ.999.) GO TO 20
                N=N+1
                IF (N.LE.MAX) U(N)=X
                IF (N.LE.MAX) GO TO 10
        N=MAX
20      CALL MXSORT(U,N)
        WRITE(3,905)
        WRITE(3,906)(U(I),I=1,N)
C       ASSIGN REPLICATIONS TO PROCESSORS
        DO 60 I=1,NT
                DO 40 L=1,MP
                B(L)=PU(L)
                IORDER(L)=L
40              CONTINUE
        CALL MNSORT(B,IORDER,MP)
        DO 50 K=1,IR
                PU(IORDER(K))=PU(IORDER(K))+U(I)
50      CONTINUE
        WRITE(3,903) I
        WRITE(3,907) (IORDER(K),K=1,IR)
        WRITE(3,908)
        WRITE(3,902) (PU(K),K=1,MP)
60      CONTINUE
901     FORMAT (//,' NUMBER OF TASKS = ',I4,//)
       *' NUMBER OF REPLICATIONS PER TASKS = ',I4,/,
       *' NUMBER OF PROCESSORS = ',I4,//)
902     FORMAT(5X,F6.4)
903     FORMAT(/,' REPLICATIONS OF TASK',I3,' ARE ASSIGNED TO PROCESSORS')
905     FORMAT(' SORTED DATA')
906     FORMAT(4X,F7.3)
907     FORMAT(4X,I4)
908     FORMAT(//,' TOTAL UTILIZATION OF PROCESSORS = ')
        STOP
        END
        SUBROUTINE MXSORT(U,N)
C       SORTS THE ARRAY U OF TASK UTILIZATION IN DECENDING ORDER
        DIMENSION U(N)
        IF (N.LE.1) RETURN
        LAST=N-1
        DO 102 I=1,LAST
                UMAX=U(I)
                JMAX=I
                JFIRST=I+1
                DO 101 J=JFIRST,N
                        IF (UMAX.GE.U(J)) GO TO 101
                        UMAX=U(J)
                        JMAX=J
101             CONTINUE
                U(JMAX)=U(I)
                U(I)=UMAX
102     CONTINUE
        RETURN
        END


        SUBROUTINE MNSORT(B,IORDER,MP)
C       SORTS THE CURRENT ARRAY OF PROCESSORS UTILIZATION IN
C       ASCENDING ORDER
        DIMENSION B(MP),IORDER(MP)
        LAST=MP-1
        DO 202 I=1,LAST
                BMIN=B(I)
                JMIN=I
                JFIRST=I+1
                DO 201 J=FIRST,MP
                        IF (BMIN.LE.B(J)) GO TO 201
                        BMIN=B(J)
                        JMIN=J
201             CONTINUE
        IDUMMY=IORDER(JMIN)
        IORDER(JMIN)=IORDER(I)
        IORDER(I)=IDUMMY
                B(JMIN)=B(I)
                B(I)=BMIN
202     CONTINUE
        RETURN
        END
```

Appendix B

The following is a listing
of the modified heuristic algorithm
discussed in Section 3.1.2.3.

```
C       THIS IS A PROGRAM FOR HEURISTIC APPROACH OF ALLCATING NT TASKS
C        WITH IR REPLICATIONS TO MP PROCESSORS SO THAT DIFFERENT
C        REPLICATIONS ARE ALLOCATED TO DISTINCT PROCESSORS AND
C        PROCESSOR UTILIZATION IS BALANCED
C       PU ARRAY KEEPS TRACK OF UTILIZATION OF PROCESSORS
C       MU ARRAY KEEPS TRACK OF MEMORY OF PROCESSORS
C       IORDER KEEPS THE ORDER OF PROCESSOR UTILIZATION ARRAY
C        WHEN SORTED IN ASCENDING ORDER
C       JORDER KEEPS THE ORDER OF TASK UTILIZATION ARRAY
C        WHEN SORTED IN DESCENDING ORDER
C       IR=NUMBER OF REPLICATIONS PER TASK
C       NT=NUMBER OF TASKS
C       MP=NUMBER OF PROCESSORS
        REAL*4 U(40),PM(12)/12*0.0/
        REAL*4 USUM,UTOTAL
        INTEGER M(40),PM(12)/12*0/
        INTEGER MSUM,MTOTAL
        INTEGER IORDER(12),JORDER(40),KORDER(5)
        READ(1,801) IR,NT
        MAX=NT
        M=0
10      READ(1,802) X,MX
                IF (X.EQ.999.) GO TO 20
                N=N+1
                JORDER(N)=N
                IF (N.LE.MAX) U(N)=X
                IF (N.LE.MAX) M(N)=MX
                IF (N.LE.MAX) GO TO 10
20      USUM=0.0
        MSUM=0
        DO 15 1=1,NT
C       ADD THE UTILIZATION AND MEMORY NEEDS OF ALL TASKS
                USUM=USUM+U(I)
                MSUM=MSUM+M(I)
15      CONTINUE
C       CALCULATE THE NUMBER OF PROCESSORS NEEDED
C       UTILIZATION CAPACITY OF A PROCESSOR = 0.5
C       MEMORY CAPACITY OF A PROCESSOR = 20000
        USUM=USUM*IR
        MSUM=MSUM*IR
        USUM=USUM/(0.5)
        MSUM=(MSUM/20000)+1
        MP=(2*USUM)
        IF (MP,IT.MSUM) MP=MSUM
        WRITE(3,901) NT,IR,MP
        CALL MXSORT(U,N,JORDER)
        WRITE(3,905)
        WRITE(3,906)(U(I),J=1,N)
C       ASSIGN I REPLICATIONS TO PROCESSORS
        DO 60 I=1,NT
                DO 40 I=1,MP
                R(I)=PU(I)
                IORDER(I)=1
40              CONTINUE
```

```
      CALL MASORT(B,IORDER,MP)
C     II = NUMBER OF REPLICATIONS ASSIGNED SO FAR AT A GIVEN STEP
C     CHECK = 1 IF MAXIMUM UTILIZATION OR CAPACITY IS ACHIEVED
C     KORDER KEEPS TRACK OF ORDER OF PROCESSORS TO WHICH
C      TASK REPLICATION ARE ASSIGNED AT A GIVEN STEP
      II=0
      DO 50 K=1,MP
              UTOTAL=PU(IORDER(K))+U(I)
              MTOTAL=PM(IORDER(K)+M(JORDER(I))
              IF ((UTOTAL.GT.(.345)).OR.(MTOTAL.GT.20000)) GO TO 50
              II=II+1
              PU(IORDER(K))=UTOTAL
              PM(IORDER(K))=MTOTAL
              KORDER(II)=IORDER(K)
              IF (II.EQ.IR) GO TO 55
50    CONTINUE
55    WRITE(3,903) I
      WRITE(3,907) (KORDER(K),K=1,IR)
      WRITE(3,908)
      WRITE(3,902) (PU(K),K=1,MP)
      WRITE(3,909)
      WRITE(3,910) (PM(K),K=1,MP)
60    CONTINUE
801   FORMAT(2I5)
802   FORMAT(F7.3,I8)
901   FORMAT (//.'NUMBER OF TASKS = ',I4,/,
     *' NUMBER OF REPLICATIONS PER TASK = ',I4,/,
     *' NUMBER OF PROCESSORS = ',I4,//)
902   FORMAT(5X,F6.4)
903   FORMAT(/,'REPLICATIONS OF TASK',I3,' ARE ASSIGNED TO PROCESSORS')
905   FORMAT(' SORTED DATA')
906   FORMAT(4X,F7.3)
907   FORMAT(4X,I4)
908   FORMAT(//,'TOTAL UTILIZATION OF PROCESSORS = ')
909   FORMAT(//,'TOTAL MEMORY OF PROCESSORS = ')
910   FORMAT(5X,I6)
      STOP
      END


      SUBROUTINE MXSORT(U,N,JORDER)
C     SORTS THE ARRAY U OF TASK UTILIZATION IN DECENDING ORDER
      DIMENSION U(N),JORDER(N)
      IF (N.LE.1) RETURN
      LAST=N-1
      DO 102 I=1,LAST
              UMAX=U(I)
              JMAX=I
              JFIRST=I+1
              DO 101 J=JFIRST,N
                      IF (UMAX.GE.U(J)) GO TO 101
                      UMAX=U(J)
                      JMAX=J
101           CONTINUE
```

```
      JDUMMY=JORDER(JMAX)
      JORDER(JMAX)=JORDER(I)
      JORDER(I)=JDUMMY
              U(JMAX)=U(I)
              U(I)=UMAX
102   CONTINUE
      RETURN
      END


      SUBROUTINE MNSORT(B,IORDER,MP)
C     SORTS THE CURRENT ARRAY OF PROCESSORS UTILIZATION IN
C     ASCENDING ORDER
      DIMENSION B(MP),IORDER(MP)
      LAST=MP
      DO 202 I=1,LAST
              BMIN=B(J)
              JMIN=I
              JFIRST=I+1
              DO 201 J=JFIRST,MP
                      IF (BMIN.LE.B(J)) GO TO 201
                      BMIN=B(J)
                      JMIN=J
201           CONTINUE
      IDUMMY=IORDER(JMIN)
      IORDER(JMIN)=IORDER(I)
      IORDER(I)=IDUMMY
              B(JMIN)=B(I)
              B(I)=BMIN
202   CONTINUE
      RETURN
      END
```

Appendix C

This appendix includes computer printouts
of the algorithms discussed in
Sections 3.1.2.2 and 3.1.2.3.

RUN #1

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 4

SORTED DATA
        0.119
        0.077
        0.069
        0.055
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1960
        0.1960
        0.0770

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
        4
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1880
        0.1960
        0.2650
        0.1460

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
        4
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.2430
        0.2510
        0.2650
        0.2010

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSORS
        4
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.2770
        0.2850
        0.2650
        0.2350

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
        4
        3
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3090
        0.2850
        0.2970
        0.2670

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
        4
        2
        3

TOTAL UTILIZATIONS OF PROCESSORS =
        0.3090
        0.3130
        0.3250
        0.2950

REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
        4
        1
        2

TOTAL UTILIZATIONS OF PROCESSORS =
        0.3320
        0.3360
        0.3250
        0.3180

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
        4
        3
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3530
        0.3360
        0.3460
        0.3390

REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
        2
        4
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3530
        0.3550
        0.3650
        0.3580

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
        1
        2
        4

TOTAL UTILIZATION OF PROCESSRS =
        0.3670
        0.3690
        0.3650
        0.3720

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
        3
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3810
        0.3830
        0.3790
        0.3720

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
```

```
        4
        3
        1
TOTAL UTILIZATION OF PROCESSORS =
        0.3900
        0.3830
        0.3080
        0.3810

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
        4
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3900
        0.3890
        0.3940
        0.3870

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
        4
        2
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3940
        0.3930
        0.3940
        0.3910

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
        4
        2
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3980
        0.3970
        0.3940
        0.3950

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        3
        4
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3980
        0.3990
        0.3960
        0.3970

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSORS
```

```
        3
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3990
        0.3990
        0.3970
        0.3980

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
        3
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.4000
        0.3990
        0.3980
        0.3990

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
        3
        4
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.4000
        0.4000
        0.3990
        0.4000

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
        3
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.4010
        0.4000
        0.4000
        0.4010

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
        2
        3
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.4010
        0.4010
        0.4010
        0.4020
```

```
REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.4020
        0.4020
        0.4020
        0.4020
```

RUN #2

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 5

SORTED DATA
        0.119
        0.077
        0.069
        0.055
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0
        0.0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        5
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1960
        0.0770
        0.0770

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
```

```
        4
        5
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.1880
        0.1190
        0.1960
        0.1460
        0.1460

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
        2
        4
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.1880
        0.1740
        0.1960
        0.2010
        0.2010

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSORS
        2
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2220
        0.2080
        0.2300
        0.2010
        0.2010

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
        4
        5
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.2220
        0.2400
        0.2300
        0.2330
        0.2330

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
        1
        3
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2500
```

```
          0.2400
          0.2580
          0.2610
          0.2330

REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
          5
          2
          1

TOTAL UTILIZATION OF PROCESSORS =
          0.2730
          0.2630
          0.2580
          0.2610
          0.2560

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
          5
          3
          4

TOTAL UTILIZATION OF PROCESSORS =
          0.2730
          0.2630
          0.2790
          0.2820
          0.2770

REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
          2
          1
          5

TOTAL UTILIZATION OF PROCESSORS =
          0.2920
          0.2820
          0.2790
          0.2820
          0.2960

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
          3
          2
          4

TOTAL UTILIZATION OF PROCESSORS =
          0.2920
          0.2960
          0.2930
          0.2960
          0.2960

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
```

```
        1
        3
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3060
        0.3100
        0.3070
        0.2960
        0.2960

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
        5
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3100
        0.3070
        0.3050
        0.3050

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
        5
        4
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3100
        0.3130
        0.3110
        0.3110

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
        2
        5
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3140
        0.3130
        0.3150
        0.3150

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
        3
        2
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3180
        0.3170
        0.3150
        0.3190

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        4
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3170
        0.3180
        0.3190
        0.3170
        0.3190

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSRS
        4
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3180
        0.3190
        0.3190
        0.3180
        0.3190

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
        4
        1
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.3190
        0.3190
        0.3190
        0.3190
        0.3200

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
        2
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3190
        0.3200
        0.3200

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
```

```
        3
        5
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3200
        0.3210
        0.3210

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
        2
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3210
        0.3210
        0.3210
        0.3210

REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        5
        2
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3220
        0.3210
        0.3220
        0.3220
```

RUN #3

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 5

SORTED DATA
        0.119
        0.077
        0.069
        0.055
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0
        0.0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        5
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1960
        0.0770
        0.0770

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
```

```
         4
         5
         1

TOTAL UTILIZATION OF PROCESSORS =
         0.1880
         0.1190
         0.1960
         0.1460
         0.1460

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
         2
         4
         5

TOTAL UTILIZATION OF PROCESSORS =
         0.1880
         0.1740
         0.1960
         0.2010
         0.2010

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSORS
         2
         1
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.2220
         0.2080
         0.2300
         0.2010
         0.2010

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
         4
         5
         2

TOTAL UTILIZATION OF PROCESSORS =
         0.2220
         0.2400
         0.2300
         0.2330
         0.2330

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
         1
         3
         4

TOTAL UTILIZATION OF PROCESSORS =
         0.2500
```

```
        0.2400
        0.2580
        0.2610
        0.2330

REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
        5
        2
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.2730
        0.2630
        0.2580
        0.2610
        0.2560

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
        5
        3
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2730
        0.2630
        0.2790
        0.2820
        0.2770

REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
        2
        1
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.2920
        0.2820
        0.2790
        0.2820
        0.2960

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
        3
        2
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2920
        0.2960
        0.2930
        0.2960
        0.2960

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
```

```
         1
         3
         2

TOTAL UTILIZATION OF PROCESSORS =
         0.3060
         0.3100
         0.3070
         0.2960
         0.2960

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
         5
         4
         1

TOTAL UTILIZATION OF PROCESSORS =
         0.3150
         0.3100
         0.3070
         0.3050
         0.3050

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
         5
         4
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.3150
         0.3100
         0.3130
         0.3110
         0.3110

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
         2
         5
         4

TOTAL UTILIZATION OF PROCESSORS =
         0.3150
         0.3140
         0.3130
         0.3150
         0.3150

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
         3
         2
         5

TOTAL UTILIZATION OF PROCESSORS =
         0.3150
```

```
        0.3180
        0.3170
        0.3150
        0.3190

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        4
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3170
        0.3180
        0.3190
        0.3170
        0.3190

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSRS
        4
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3180
        0.3190
        0.3190
        0.3180
        0.3190

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
        4
        1
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.3190
        0.3190
        0.3190
        0.3190
        0.3200

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
        2
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3190
        0.3200
        0.3200

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
```

```
        3
        5
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3200
        0.3210
        0.3210

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
        2
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3210
        0.3210
        0.3210
        0.3210

REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        5
        2
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3220
        0.3210
        0.3220
        0.3220
```

RUN #4

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 5

SORTED DATA
        0.119
        0.077
        0.069
        0.059
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0
        0.0

TOTAL MEMORY OF PROCESSORS =
        1500
        1500
        1500
           0
           0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        5
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1960
        0.0770
        0.0770

TOTAL MEMORY OF PROCESSORS =
        1500
        1500
        2810
        1310
        1310

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
        4
        5
        1

TOTAL UTILIZATION OF PROCESSRS =
        0.1800
        0.1190
        0.1960
        0.1460
        0.1460

TOTAL MEMORY OF PROCESSRS =
        1592
        1500
        2810
        1402
        1402

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
        2
        4
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.1880
        0.1740
        0.1960
        0.2010
        0.2010

TOTAL MEMORY OF PROCESSORS =

        1592
        2525
        2810
        2427
        2427

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSORS
        2
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2220
        0.2080
        0.2800
        0.2010
        0.2010

TOTAL MEMORY OF PROCESSORS =
        3842
        4779
        5060
        2427
        2427

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
        4
        5
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.2220
        0.2400
        0.2300
        0.2330
        0.2330

TOTAL MEMORY OF PROCESSORS =
        3842
       11025
        5060
        8677
        8677

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
        1
        3
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2500
        0.2400
        0.2580
        0.2610
        0.2330

TOTAL MEMORY OF PROCESSORS =
        4392
       11025
        5616
        9227
        8677
```

```
REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
        5
        2
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.2730
        0.2630
        0.2580
        0.2610
        0.2560

TOTAL MEMORY OF PROCESSORS =
        6467
       13100
        5610
        9227
       10752

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
        5
        3
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2730
        0.2630
        0.2790
        0.2820
        0.2770

TOTAL MEMORY OF PROCESSORS =
        6467
       13100
        6810
       10427
       11952

REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
        2
        1
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.2920
        0.2820
        0.2790
        0.2820
        0.2960

TOTAL MEMORY OF PROCESSORS =
       15807
       22440
        6810
       10427
       21292

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
        3
        2
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.2920
        0.2960
        0.2930
        0.2960
        0.2960

TOTAL MEMORY OF PROCESSORS =
       15807
       22500
        6876
       10487
       21292

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
        1
        3
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3060
        0.3100
        0.3070
        0.2960
        0.2960

TOTAL MEMORY OF PROCESSORS =
       17707
       24400
        8770
       10487
       21292

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
        5
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3100
        0.3076
        0.3050
        0.3050

TOTAL MEMORY OF PROCESSORS =
        18137
        24400
         8770
        10917
        21722

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
        5
        4
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3100
        0.3130
        0.3110
        0.3110

TOTAL MEMORY OF PROCESSORS =
        18137
        24400
         9388
        11527
        22332

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
        2
        5
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3140
        0.3130
        0.3150
        0.3150

TOTAL MEMORY OF PROCESSORS =
        18137
        24905
         9380
        12032
        22837

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
        3
        2
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.3150
        0.3180
        0.3170
        0.3150
        0.3190

TOTAL MEMORY OF PROCESSORS =
        18137
        25205
         9680
        12032
        23137

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        4
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3170
        0.3180
        0.3190
        0.3170
        0.3190

TOTAL MEMORY OF PROCESSORS =
        19437
        25205
        10980
        13332
        23137

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSORS
        4
        1
        2

TOTAL UTILIZATION OF PROCESSORS =
        0.3180
        0.3190
        0.3190
        0.3190
        0.3180
        0.3190

TOTAL MEMORY OF PROCESSORS =
        19999
        25767
        10980
        13894
        23137

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
        4
        1
        5

TOTAL UTILIZATION OF PROCESSORS =
        0.3190
        0.3190
        0.3190
        0.3190
        0.3200

TOTAL MEMORY OF PROCESSORS =
        20314
        25767
        10980
        14209
        23452

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
        2
        4
        1

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3190
        0.3200
        0.3200

TOTAL MEMORY OF PROCESSORS =
        20564
        26017
        10980
        14459
        23452

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
        3
        5
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3200
        0.3200
        0.3200
        0.3210
        0.3210

TOTAL MEMORY OF PROCESSORS =
        20564
        26017
        11980
        15459
        24452

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
        2
        1
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3210
        0.3210
        0.3210
        0.3210

TOTAL MEMORY OF PROCESSORS =
        21564
        27017
        12980
        15459
        24452

REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        5
        2
        4

TOTAL UTILIZATION OF PROCESSORS =
        0.3210
        0.3220
        0.3210
        0.3220
        0.3220

TOTAL MEMORY OF PROCESSORS =
        21564
        27152
        12980
        15594
        24587
```

RUN #5

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 6

SORTED DATA
        0.119
        0.077
        0.069
        0.055
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0
        0.0
        0.0

TOTAL MEMORY OF PROCESSORS =
        1500
        1500
        1500
           0
           0
           0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        5
        6
```

```
TOTAL UTILIZATION OF PROCESSORS =
         0.1190
         0.1190
         0.1190
         0.0770
         0.0770
         0.0770

TOTAL MEMORY OF PROCESSORS =
         1500
         1500
         1500
         1310
         1310
         1310

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
         4
         5
         6

TOTAL UTILIZATION OF PROCESSORS =
         0.1190
         0.1190
         0.1190
         0.1460
         0.1460
         0.1460

TOTAL MEMORY OF PROCESSRS =
         1500
         1500
         1500
         1402
         1402
         1402

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.1740
         0.1740
         0.1740
         0.1460
         0.1460
         0.1460

TOTAL MEMORY OF PROCESSORS =
         2525
         2525
         2525
```

```
        1402
        1402
        1402

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSRS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.1740
        0.1740
        0.1740
        0.1800
        0.1800
        0.1800

TOTAL MEMORY OF PROCESSORS =
        2525
        2525
        2525
        3652
        3652
        3652

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2060
        0.2060
        0.2060
        0.1800
        0.1800
        0.1800

TOTAL MEMORY OF PROCESSORS =
        8775
        8775
        8775
        3652
        3652
        3652

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2060
        0.2060
        0.2060
        0.2080
        0.2080
        0.2080

TOTAL MEMORY OF PROCESSORS =
        8775
        8775
        8775
        4202
        4202
        4202

REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2290
        0.2290
        0.2290
        0.2080
        0.2080
        0.2080

TOTAL MEMORY OF PROCESSORS =
        10850
        10850
        10850
        4202
        4202
        4202

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2290
        0.2290
        0.2290
        0.2290
        0.2290
        0.2290

TOTAL MEMORY OF PROCESSORS =
        10850
        10850
        10850
        5402
        5402
        5402
```

```
REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2480
        0.2480
        0.2480
        0.2290
        0.2290
        0.2290

TOTAL MEMORY OF PROCESSORS =
        20190
        20190
        20190
        5402
        5402
        5402

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2480
        0.2480
        0.2480
        0.2430
        0.2430
        0.2430

TOTAL MEMORY OF PROCESSORS =
        20190
        20190
        20190
         5462
         5462
         5462

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2480
        0.2480
        0.2480
        0.2570
        0.2570
        0.2570
```

```
TOTAL MEMORY OF PROCESSORS =
        20190
        20190
        20190
         7362
         7362
         7362

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2570
        0.2570
        0.2570
        0.2570
        0.2570
        0.2570

TOTAL MEMORY OF PROCESSORS =
        20620
        20620
        20620
         7362
         7362
         7362

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2570
        0.2570
        0.2570

TOTAL MEMORY OF PROCESSORS =
        21230
        21230
        21230
         7362
         7362
         7362

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
        4
        5
        6
```

```
TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2610
        0.2610
        0.2610

TOTAL MEMORY OF PROCESSORS =
        21230
        21230
        21230
         7867
         7867
         7867

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2650
        0.2650
        0.2650

TOTAL MEMORY OF PROCESSORS =
        21230
        21230
        21230
         8167
         8167
         8167

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2650
        0.2650
        0.2650
        0.2650
        0.2650
        0.2650

TOTAL MEMORY OF PROCESSORS =
        22530
        22530
        22530
```

```
          8167
          8167
          8167

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
        0.2660
        0.2660
        0.2660
        0.2650
        0.2650
        0.2650

TOTAL MEMORY OF PROCESSORS =
        23092
        23092
        23092
         8167
         8167
         8167

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
         4
         5
         6

TOTAL UTILIZATION OF PROCESSORS =
        0.2660
        0.2660
        0.2660
        0.2660
        0.2660
        0.2660

TOTAL MEMORY OF PROCESSORS =
        23092
        23092
        23092
         8482
         8482
         8482

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
        0.2670
        0.2670
```

```
         0.2670
         0.2660
         0.2660
         0.2660

TOTAL MEMORY OF PROCESSORS =
         23342
         23342
         23342
          8482
          8482
          8482

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
         4
         5
         6

TOTAL UTILIZATION OF PROCESSORS =
         0.2670
         0.2670
         0.2670
         0.2670
         0.2670
         0.2670

TOTAL MEMORY OF PROCESSORS =
         23342
         23342
         23342
          9482
          9482
          9482

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.2680
         0.2680
         0.2680
         0.2670
         0.2670
         0.2670

TOTAL MEMORY OF PROCESSORS =
         24342
         24342
         24342
          9482
          9482
          9482
```

```
REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2680
        0.2680
        0.2680
        0.2680
        0.2680
        0.2680

TOTAL MEMORY OF PROCESSORS =
        24342
        24342
        24342
         9617
         9617
         9617
```

RUN #6

```
NUMBER OF TASKS = 23
NUMBER OF REPLICATIONS PER TASK = 3
NUMBER OF PROCESSORS = 6

SORTED DATA
        0.119
        0.077
        0.069
        0.055
        0.034
        0.032
        0.028
        0.023
        0.021
        0.019
        0.014
        0.014
        0.009
        0.006
        0.004
        0.004
        0.002
        0.001
        0.001
        0.001
        0.001
        0.001
        0.001

REPLICATIONS OF TASK 1 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0
        0.0
        0.0

TOTAL MEMORY OF PROCESSORS =
          1500
          1500
          1500
             0
             0
             0

REPLICATIONS OF TASK 2 ARE ASSIGNED TO PROCESSORS
        4
        5
        6
```

```
TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.0770
        0.0770
        0.0770

TOTAL MEMORY OF PROCESSORS =
        1500
        1500
        1500
        1310
        1310
        1310

REPLICATIONS OF TASK 3 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.1190
        0.1190
        0.1190
        0.1460
        0.1460
        0.1460

TOTAL MEMORY OF PROCESSORS =
        1500
        1500
        1500
        1402
        1402
        1402

REPLICATIONS OF TASK 4 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.1740
        0.1740
        0.1740
        0.1460
        0.1460
        0.1460

TOTAL MEMORY OF PROCESSORS =
        2525
        2525
        2525
        1402
        1402
        1402

REPLICATIONS OF TASK 5 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.1740
        0.1740
        0.1740
        0.1800
        0.1800
        0.1800

TOTAL MEMORY OF PROCESSORS =
        2525
        2525
        2525
        3652
        3652
        3652

REPLICATIONS OF TASK 6 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2060
        0.2060
        0.2060
        0.1800
        0.1800
        0.1800

TOTAL MEMORY OF PROCESSORS =
        8775
        8775
        8775
        3652
        3652
        3652

REPLICATIONS OF TASK 7 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2060
        0.2060
        0.2060
        0.2080
        0.2080
        0.2080

TOTAL MEMORY OF PROCESSORS =
        8775
        8775
        8775
        4202
        4202
        4202

REPLICATIONS OF TASK 8 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2290
        0.2290
        0.2290
        0.2080
        0.2080
        0.2080

TOTAL MEMORY OF PROCESSORS =
        10850
        10850
        10850
         4202
         4202
         4202

REPLICATIONS OF TASK 9 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2290
        0.2290
        0.2290
        0.2290
        0.2290
        0.2290

TOTAL MEMORY OF PROCESSORS =
        10850
        10850
        10850
         5402
         5402
         5402
```

```
REPLICATIONS OF TASK 10 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2290
        0.2290
        0.2290
        0.2480
        0.2480
        0.2480

TOTAL MEMORY OF PROCESSORS =
        10850
        10850
        10850
        14742
        14742
        14742

REPLICATIONS OF TASK 11 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2430
        0.2430
        0.2430
        0.2480
        0.2480
        0.2480

TOTAL MEMORY OF PROCESSORS =
        10910
        10910
        10910
        14742
        14742
        14742

REPLICATIONS OF TASK 12 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2570
        0.2570
        0.2570
        0.2480
        0.2480
        0.2480
```

```
TOTAL MEMORY OF PROCESSORS =
        12810
        12810
        12810
        14742
        14742
        14742

REPLICATIONS OF TASK 13 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2570
        0.2570
        0.2570
        0.2570
        0.2570
        0.2570

TOTAL MEMORY OF PROCESSORS =
        12810
        12810
        12810
        15172
        15172
        15172

REPLICATIONS OF TASK 14 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2570
        0.2570
        0.2570

TOTAL MEMORY OF PROCESSORS =
        13420
        13420
        13420
        15172
        15172
        15172

REPLICATIONS OF TASK 15 ARE ASSIGNED TO PROCESSORS
        4
        5
        6
```

```
TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2610
        0.2610
        0.2610

TOTAL MEMORY OF PROCESSORS =
        13420
        13420
        13420
        15677
        15677
        15677

REPLICATIONS OF TASK 16 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2630
        0.2630
        0.2630
        0.2650
        0.2650
        0.2650

TOTAL MEMORY OF PROCESSORS =
        13420
        13420
        13420
        15977
        15977
        15977

REPLICATIONS OF TASK 17 ARE ASSIGNED TO PROCESSORS
        1
        2
        3

TOTAL UTILIZATION OF PROCESSORS =
        0.2650
        0.2650
        0.2650
        0.2650
        0.2650
        0.2650

TOTAL MEMORY OF PROCESSRS =
        14720
        14720
        14720
```

```
         15977
         15977
         15977

REPLICATIONS OF TASK 18 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.2660
         0.2660
         0.2660
         0.2650
         0.2650
         0.2650

TOTAL MEMORY OF PROCESSORS =
         15282
         15282
         15282
         15977
         15977
         15977

REPLICATIONS OF TASK 19 ARE ASSIGNED TO PROCESSORS
         4
         5
         6

TOTAL UTILIZATION OF PROCESSORS =
         0.2660
         0.2660
         0.2660
         0.2660
         0.2660
         0.2660

TOTAL MEMORY OF PROCESSORS =
         15282
         15282
         15282
         16292
         16292
         16292

REPLICATIONS OF TASK 20 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.2670
         0.2670
         0.2670
         0.2660
         0.2660
         0.2660

TOTAL MEMORY OF PROCESSORS =
         15532
         15532
         15532
         16292
         16292
         16292

REPLICATIONS OF TASK 21 ARE ASSIGNED TO PROCESSORS
         4
         5
         6

TOTAL UTILIZATION OF PROCESSORS =
         0.2670
         0.2670
         0.2670
         0.2670
         0.2670
         0.2670

TOTAL MEMORY OF PROCESSORS =
         15532
         15532
         15532
         17292
         17292
         17292

REPLICATIONS OF TASK 22 ARE ASSIGNED TO PROCESSORS
         1
         2
         3

TOTAL UTILIZATION OF PROCESSORS =
         0.2680
         0.2680
         0.2680
         0.2670
         0.2670
         0.2670

TOTAL MEMORY OF PROCESSORS =
         16532
         16532
         16532
         17292
         17292
         17292
```

```
REPLICATIONS OF TASK 23 ARE ASSIGNED TO PROCESSORS
        4
        5
        6

TOTAL UTILIZATION OF PROCESSORS =
        0.2680
        0.2680
        0.2680
        0.2680
        0.2680
        0.2680

TOTAL MEMORY OF PROCESSORS =
        16532
        16532
        16532
        17427
        17427
        17427
```

1. Report No. 165926	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle PROBLEMS RELATED TO THE INTEGRATION OF FAULT-TOLERANT AIRCRAFT ELECTRONIC SYSTEMS		5. Report Date June 1982
		6. Performing Organization Code
7. Author(s) J. A. Bannister, K. Trivedi, V. Adlakha, and T. A. Alspaugh, Jr.		8. Performing Organization Report No. 505-34-43-06
9. Performing Organization Name and Address Systems and Measurements Division Research Triangle Institute P.O. Box 12194 Research Triangle Park, NC 27709		10. Work Unit No.
		11. Contract or Grant No. NAS1-16489
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This report explores several problems related to the design of integrated fault-tolerant aircraft electronic systems.

Problems related to the design of the hardware for an integrated aircraft electronic system are first considered. Taxonomies of concurrent systems are reviewed and a new taxonomy is proposed. An informal methodology intended to identify feasible regions of the taxonomic design space is described. Specific tools are recommended for use in the methodology. Based on the methodology, a preliminary "strawman" integrated fault-tolerant aircraft electronic system is proposed.

Next, problems related to the programming and control of integrated aircraft electronic systems are discussed. Issues of system resource management, including the scheduling and allocation of real-time periodic tasks in a multiprocessor environment, are treated in detail. The role of software design in integrated fault-tolerant aircraft electronic systems is discussed.

Conclusions and recommendations for further work are included.

17. Key Words (Suggested by Author(s)) design methodology, fault-tolerance integrated aircraft electronic systems, multiprocessor taxonomy, real-time periodic scheduling, reliability, software engineering, task allocation		18. Distribution Statement Unclassified - Unlimited Subject Category 61	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 168	22. Price*